Lecture notes for MAE 207: Finite element analysis for coupled problems

David Kamensky

Fall 2022

Preface

These notes represent an experiment in presenting rigorous numerical analysis of finite element methods for linear partial differential equation (PDE) systems alongside a more practical approach to nonlinear multiphysics problems. While our presentation is significantly more abstract than most treatments of the latter, we leverage recent advances in code generation to connect a high-level mathematical perspective directly to computations, while bypassing much of the discussion on indexing degrees of freedom, mapping between reference and physical elements, etc., which can dominate many practical courses on finite element analysis (FEA).

This material is intended to be covered in full within a single graduate-level course for engineering students. The content is split into two parts. Part I builds up the language in which we discuss FEA, making liberal use of concepts from functional analysis, reviewed in Chapter 1. While the initial reaction of some students to this is that it is "too mathematical" or "too abstract", classroom experience indicates that engineering students absorb the material readily at the level needed to continue, i.e., basic proficiency in the definitions of terms and how they relate to PDE theory. We emphasize only the operational aspects needed to follow the contours of *a priori* error analysis (Chapter 2) and understand the user interface of modern code generation systems for FEA of arbitrary PDE systems (Chapter 3); detailed proofs of functional analysis theorems are largely passed over in favor of applying them to FEA. Part II puts these principles into practice on a representative example of a multiphysics problem, namely, fluid–structure interaction (FSI). The goal is not to provide a comprehensive course on FSI (as done, e.g., in [1]), but rather to illustrate ideas that can be extended to variational discretizations of other complex coupled PDE systems. In a typical cohort of graduate engineering students, this may be the first exposure some have to rigorous treatment of large-deformation kinematics; experience indicates that students find it to be the most challenging aspect of the course, even if it initially appears more familiar and concrete than functional analysis, due to its grounding in a specific physical application.

To connect our abstract presentation of FEA to practical calculation, we rely heavily on code generation, as alluded above. Specifically, computational examples and exercises use open-source software from the FEniCS Project [2]. Code for examples and exercises is maintained in an open-source repository on the development platform GitHub [3]. A pedagogical disadvantage of code generation is that it hides the underlying data structures and algorithms behind FEA, which we still believe are important knowledge for expert analysts. Thus, it is recommended for students to also take a more standard course in FEA, following, e.g., [4], but that is not a strict prerequisite for the present material. On the other hand, FEniCS's abstract perspective of variational forms and function spaces rather than elements and nodes applies equally to numerical methods that go beyond traditional FEA, such as emerging meshfree and isogeometric methods, which have recently become available in some commercial engineering software.

Notes for Fall 2022

This section itemizes a few points to keep in mind about the Fall 2022 version of these notes.

- For Fall 2022, code examples will continue to be implemented using “Legacy FEniCS” (version 2019.2). **It is now recommended that new FEniCS-based research projects use the FEniCSx redesign**, but its application programming interface (API) continues to be updated regularly and most supplemental resources online still use the legacy API, which may lead to confusion in a classroom setting. In any case, students should focus on the mathematical foundations of FEA and the FEniCS philosophy of automated code generation; these aspects remain unchanged in FEniCSx and are often at the root of users’ difficulties applying FEniCS(x).
- These notes are no longer under active development, as the author left academia and the course is not offered anymore. However, readers should feel free to suggest corrections or improvements as issues via the corresponding repository of code examples [3].

Contents

I

Physics-neutral finite elements

The computer speaks mathematics, not physics.
—Leszek F. Demkowicz

Chapter 1

Mathematical background: Functional analysis and partial differential equations

1.1 Introduction

To approach the topic of finite element analysis (FEA) for general systems of coupled partial differential equations (PDEs), spanning multiple scientific disciplines, we need a one-size-fits-all physics-independent language for describing PDE systems and our approximations of them. In the preface to his famous textbook on PDEs [5], L. C. Evans writes, in boldface font, that "PDE theory is not a branch of functional analysis". However, that he considered this clarification necessary is indicative of the fundamental role played by functional analysis in the study of PDEs. In the context of FEA, it is particularly useful to view PDEs through the lens of functional analysis, which provides a general framework for developing finite element methods to approximate different (combinations of) PDE systems. The practical power of this viewpoint is perhaps nowhere more evident than in the rise of code generation systems, like FEniCS, which can automatically compile high-performance FEA routines from concise problem statements written in the abstract language of function spaces and variational forms.

However, fully harnessing the power of functional analysis for FEA comes with an up-front learning cost. Both functional analysis and PDE theory are challenging branches of mathematics, each requiring years of graduate-level study to master. The purpose of this chapter is not to provide comprehensive or rigorous coverage, but simply to define enough notation and jargon to make a modern presentation and automated implementation of FEA intelligible. This brief overview cannot replace full course sequences on real analysis, functional analysis, and PDE theory. However, coursework in the aforementioned areas is *not* a prerequisite.

We assume that the reader is familiar with basic notations of logic and set theory, well-practiced in the operational aspects of multivariate calculus,[1] and acquainted with the concept of a PDE. Certain results of fundamental mathematical significance, like the Hahn–Banach theorem (which would appear on any mathematician's "short list" of functional analysis topics), are completely omitted, as they are not needed to follow—at least superficially—the convergence analysis of finite element methods. To borrow a phrase from T. J. R. Hughes, an advanced understanding of finite

[1]Readers encountering difficulty with the various "integration by parts" identities used freely in this course should complete the optional review exercises in Appendix A.

element methods requires analysts to "know the moves" of functional analysis, but not necessarily all of their rigorous justifications. Important pieces of **jargon** are highlighted in bold, near their definitions.

1.2 Topology

To approach numerical approximation rationally, and have a precise notion of a given method's accuracy, we need a mathematical framework for quantifying how close functions are to one another. This turns out to be more complicated than talking about how close, say, points in $\mathbb{R}^n$ are to each other, for fundamental reasons. To understand why, we begin by introducing some mathematical abstractions that give us a suitably-general (but still precise) concept of "closeness". These abstractions, which underpin the mathematical subject of topology, remain applicable to $\mathbb{R}^n$, and are also commonly invoked when describing the domains over which PDEs are posed.

1.2.1 Abstract topology

The most general expression of interconnectedness of a set X (e.g., of functions we want to compare with each other) is a **topology**, τ, which is itself a collection of so-called **open subsets** of X, such that

- The empty set and X are open: $\emptyset \in \tau \ \land \ X \in \tau$.
- All unions of open sets are open: $(\bigcup_{x \in A} x) \in \tau$ for any (possibly infinite) collection $A \subset \tau$.
- Binary intersections are open: $A \in \tau \land B \in \tau \ \Rightarrow \ A \cap B \in \tau$, which extends trivially, by induction, to any finite intersection.

The complement of any open set in τ is referred to as a **closed set**.

1.2.2 Metric topology

While the above definitions are very general, it is usually the case in applications that topologies are **induced** by some more specific structure. In particular, many scenarios of interest in numerical analysis involve metric topologies. A **metric** is a function $d : X \times X \to \mathbb{R}$ which is interpreted as returning the distance between its two arguments. The properties it must satisfy are:

- Positivity: $\forall x, y \in X \ , \ d(x, y) \geq 0 \ \land \ d(x, y) = 0 \iff x = y$.
- Symmetry: $\forall x, y \in X \ , \ d(x, y) = d(y, x)$.
- The **triangle inequality**: $\forall x, y, z \in X \ , \ d(x, y) \leq d(x, z) + d(z, y)$.

The mechanism by which a metric induces a topology is as follows: $U \subset X$ is open if, $\forall x \in U$, $\exists r > 0$ such that $B_r(x) \subset U$, where $B_r(x)$ is the **open ball** of radius r centered at x, viz., $B_r(x) := \{y \in X \ : \ d(x, y) < r\}$.

1.2.3 Useful topological concepts

In numerical analysis, one frequently encounters a number of topological concepts, both when considering the topology of function spaces and the topology of the domains in $\mathbb{R}^n$ over which PDE problems are posed.

Convergence

When analyzing numerical methods, we are usually interested in knowing when the distances between functions are converging to zero as some discrete approximation scheme is refined. A sequence $\{x_n\}_{n=1}^{\infty} \subset X$ is said to **converge** to $x \in X$ if any open set containing x also contains all but finitely-many elements of the sequence. In metric spaces, this has a more concrete manifestation: $\forall \epsilon > 0$, $\exists N$ such that, $\forall m > N$, $d(x_m, x) < \epsilon$. A sequence is **Cauchy** if its elements are all getting closer together, i.e., $\forall \epsilon > 0$, $\exists N$ such that, $\forall m, n > N$, $d(x_n, x_m) < \epsilon$. It may be counter-intuitive, but being Cauchy *does not* guarantee that there is some x to which the sequence converges. A metric space in which every Cauchy sequence converges to some limit is described as **complete**.

Continuity

Abstractly, a mapping $f : X \to Y$, between topological spaces X and Y is **continuous**, if the inverse image of every open set from Y is open in X. For metric topologies, we have the more intuitive characterization that, f is continuous at $x \in X$ if, for every $\epsilon > 0$, there exists δ such that $f(B_\delta(x)) \subset B_\epsilon(f(x))$. If δ is independent of x, then f is said to be **uniformly continuous**.

Limit points, closure, and density

A **limit point** of $U \subset X$ is a point $x \in X$ such that every open set $O \in \tau$ containing x has a non-empty intersection with U. Alternatively, it is characterized by the existence of a sequence $\{x_n\}_{n=1}^{\infty} \subset U$ that converges to it. The union of U with the set of its limit points is closed and referred to as the **closure** of U, denoted $\overline{U}$. A concept dual to the closure of U is its **interior**, $U^\circ := (\overline{U^c})^c$, where the superscript c indicates the set complement. Closure and interior give rise to the idea of a set **boundary** of U, $\partial U := \overline{U} \setminus U^\circ$.

If every point in X is either in U or a limit point of U, then it is said that U is **dense** in X. It is often easier to show some property or state some definition on a dense subset of the desired space X, where the dense subset satisfies nicer properties. For instance, in PDE theory, it is more convenient to do calculus manipulations on smooth functions, where all derivatives are well-defined, but we might like the results of these manipulations to apply to less smooth collections of functions. Suppose a subset U is dense in metric space X. If a mapping $f : U \to Y$ is uniformly continuous, and Y is a complete metric space, then we can **extend** f to the uniformly-continuous mapping $\tilde{f} : X \to Y$ in a unique way. In particular, density implies that any point x in $X \setminus U$ is the limit of some $\{x_n\} \subset U$, and this extension is defined by $\tilde{f}(x) = \lim_{n\to\infty} f(x_n)$.

1.3 Linear algebra

The topological concepts of Section 1.2 apply to all sorts of collections of objects. However, when considering collections of functions, we often also have a very useful algebraic structure: **linearity**. A **linear space** is a set V that is closed under linear combination, i.e.,

$$\alpha v + \beta u \in V \qquad \forall v, u \in V \ , \ \forall \alpha, \beta \in \mathbb{F} \ , \tag{1.1}$$

where $\mathbb{F}$ is some field of scalars, usually $\mathbb{R}$ or $\mathbb{C}$. A complete and rigorous definition includes further properties, although these merely stipulate that addition and scalar multiplication behave in the ways we are used to. If a subset U of V is itself a linear space, then U is known as a **subspace** of V.

1.3.1 Basis and dimensionality

A critical concept in linear algebra is dimensionality. The **dimension** of a linear space is the cardinality of the smallest **Hamel basis** $B \subset V$ such that every element of V can be expressed as a linear combination of elements from B. In many practical examples of linear spaces, B is a finite set, e.g., the standard basis $\{\mathbf{e}_1, \cdots, \mathbf{e}_n\}$ of $\mathbb{R}^n$, and such spaces are therefore **finite dimensional**. However, consider, for example, the case of continuous, bounded functions. These form a linear space, in the sense that any linear combination of such function is again continuous and bounded. However, given any finite set of continuous bounded functions, one can always come up with some new function that is not a linear combination of those. As such, continuous and bounded functions form an **infinte dimensional** linear space.

Remark 1. Often the term "basis", with no qualifier, is understood to refer to the strictly algebraic concept of a Hamel basis, as discussed above. However, other topological concepts of basis may be useful when considering linear spaces with topological structure, as we do in Section 1.4.

1.3.2 Linear mappings

A mapping $A : U \to V$, for U and V both linear spaces, is referred to as a **linear mapping** if it commutes with linear combination, i.e.,

$$A(\alpha u + \beta v) = \alpha A(u) + \beta A(v) \ . \tag{1.2}$$

The set of all linear mappings $f : V \to \mathbb{F}$, where the range space is the scalar field, is called the **algebraic dual space** to V and denoted V^*. Mappings whose range is scalar are also referred to as **forms**. Frequently, the application of a linear form $f \in V^*$ to an element $v \in V$ is written in a notation called **duality pairing**:

$${}_{V^*}\langle f, v\rangle_V := f(v) \ , \tag{1.3}$$

where the subscripts on the left-hand side may be omitted if there is no risk of confusion. The **transpose** of a linear map $A : U \to V$ is denoted $A^T : V^* \to U^*$ and defined such that

$${}_{V^*}\langle f, A(v)\rangle_V = {}_{U^*}\langle A^T(f), v\rangle_U \ . \tag{1.4}$$

A mapping with more arguments may be linear in some or all of them, but a particular case of interest is a **bilinear form**, $a : V \times V \to \mathbb{F}$ such that both $a(u, \cdot)$ and $a(\cdot, u)$ are linear $\forall u \in V$. We will also encounter **semilinear mappings**, for which only one of $a(u, \cdot)$ or $a(\cdot, u)$ is linear.

1.3.3 Inner products

A particularly important class of semilinear forms are **inner products**. A semilinear form $(\cdot,\cdot) : V \times V \to \mathbb{F}$ is an inner product if it satisfies

- $(\cdot, u)$ is linear $\forall u \in V$.
- (Conjugate) symmetry: $(a, b) = \overline{(b, a)}$, where the overline denotes complex conjugation, when $\mathbb{F} = \mathbb{C}$. Considering $b = a$, this implies that $(a, a) \in \mathbb{R}\ \forall a \in V$, even if $\mathbb{F} = \mathbb{C}$.
- Positive definiteness: $\forall a \in V\ ,\ (a, a) \geq 0\ \wedge\ (a, a) = 0 \iff a = 0$.

These properties imply **conjugate linearity** of $(u, \cdot)$, $\forall u \in V$, i.e., $(a, \alpha b) = \overline{\alpha}(a, b)$, $\forall a, b \in V$ and $\forall \alpha \in \mathbb{F}$. Note that when $\mathbb{F} = \mathbb{R}$, $(\cdot,\cdot)$ is bilinear and symmetric. When a linear space is paired with an inner product, it is referred to as an **inner product space**. A familiar example is $\mathbb{R}^n$ equipped with the dot product. The significance of inner products will become apparent in Section 1.4, as they are one of the most useful tools for introducing topology to linear spaces.

1.4 Functional analysis

The study of linear spaces endowed with topological structure is called **functional analysis**. A simple example of such a space would be $\mathbb{R}^n$ with Euclidean distance acting as a metric. It turns out that any finite dimensional linear space can be mapped onto that example without losing any important structure, by taking coordinates in a given basis and applying the notion of "norm equivalence" defined below. Where functional analysis becomes interesting is in infinite dimensional spaces. The main examples of infinite dimensional linear spaces appearing in applications are sets of functions, which is why the subject is called functional analysis.

1.4.1 Normed linear spaces

The common kind of topology applied to linear spaces is a type of metric topology induced by a **norm**. A norm is a function $||\cdot|| : V \to \mathbb{R}$ such that satisfies

- Positive definiteness: $\forall u \in V\ ,\ ||u|| \geq 0\ \wedge\ ||u|| = 0 \iff u = 0$.
- Linear scaling: $\forall u \in V, \forall \alpha \in \mathbb{F}\ ,\ ||\alpha u|| = |\alpha| ||u||$, where $|\cdot|$ is absolute value when $\mathbb{F} = \mathbb{R}$ or modulus when $\mathbb{F} = \mathbb{C}$.
- Triangle inequality: $\forall u, v \in V\ ,\ ||u + v|| \leq ||u|| + ||v||$.

The mapping defined by

$$d(u, v) := ||v - u||\ , \tag{1.5}$$

may easily be shown to satisfy the properties of a metric and therefore induce a topology on V (Exercise 1). A linear space equipped with a norm is known as a **normed linear space**. A **seminorm** $|\cdot| : V \to \mathbb{R}$ satisfies the same properties as a norm, with the exception of the condition $|u| = 0 \iff u = 0$, so $|u|$ may be zero for nonzero u. If a normed linear space is complete, then it is referred to as a **Banach space**. All finite-dimensional normed linear spaces are Banach spaces.

Two norms $||\cdot||_1$ and $||\cdot||_2$ are considered **equivalent norms** if $\exists C > 0$ such that $\forall u \in V$

$$C||u||_1 \leq ||u||_2 \leq \frac{1}{C}||u||_1 \ , \tag{1.6}$$

i.e., the two norms are bounded above and below by one another, up to constant factors. In particular, if a sequence converges in $||\cdot||_1$, it also converges in $||\cdot||_2$, and vice versa. Equivalent norms induce identical topologies. If V is finite dimensional, then all valid norms are equivalent, but this is not true of infinite dimensional spaces.

Norms $||\cdot||_U$ and $||\cdot||_V$ on U and V induce a corresponding norm on the space of linear mappings from V to U:

$$||f||_{V,U} := \sup_{\substack{v\in V\\ v\neq 0}} \frac{||f(v)||_U}{||v||_V} \ , \tag{1.7}$$

for a linear map $f : V \to U$. The notation "$\sup_{x\in A} F(x)$" is an abbreviation for the **supremum** of F over A, meaning, the smallest upper bound of some real-valued function F restricted to arguments in A. The corresponding notation for the largest lower bound, the **infimum**, is "inf".[2]

A linear map that is bounded in the norm (1.7) is also continuous, and all continuous linear mappings will be bounded. The set of all continuous linear functionals (i.e., $U = \mathbb{F}$) is the **topological dual space** of V, which, for infinite-dimensional spaces, may differ from the algebraic dual introduced earlier, but the distinction is often obscured in the literature by calling both the "dual space", and denoting both by V^*. Unless otherwise specified, "dual space" and the V^* notation will always refer to the *topological* dual space.

1.4.2 Inner product spaces

A specific type of norm that is frequently encountered is one induced by an inner product. The norm defined by

$$||u|| := \sqrt{(u,u)} \tag{1.8}$$

satisfies the necessary properties when $(\cdot,\cdot)$ is an inner product (Exercise 2). In the case of $\mathbb{R}^n$ equipped with the dot product, the induced norm is simply Euclidean distance. One of the most useful properties relating inner products and their induced norms is the **Cauchy–Schwarz inequality**, which states that $\forall u, v \in V$,

$$|(u,v)| \leq ||u||\,||v|| \ . \tag{1.9}$$

In the analysis of some types of finite element methods, the Cauchy–Schwartz inequality is frequently chained together with **Young's inequality**,

$$ab \leq \frac{a^2}{2\epsilon} + \frac{\epsilon b^2}{2} \quad \forall a, b \in \mathbb{R}, \forall \epsilon > 0 \ , \tag{1.10}$$

taking a and b as $||u||$ and $||v||$.

[2] It is sometimes useful to heuristically think of inf and sup as min and max, although these are not always equivalent, e.g., $\inf_{x\in(0,1)} x$ is zero, even though zero is not in the open interval $(0,1)$, and therefore cannot be the minimum (which, in this case, is not defined).

1.4.3 Hilbert spaces

An inner product space that is complete in its induced norm topology is referred to as a **Hilbert space**, which is the most commonly-encountered type of Banach space in the analysis of PDEs and numerical methods.

Remark 2. All finite dimensional inner product spaces are Hilbert spaces.

If an inner product space is Hilbert, this carries with it implications for interpreting its dual. In particular, for every element $v \in V$, there exists a corresponding functional $R_V(v) \in V^*$ such that

$$\langle R_V(v), u \rangle = (u, v) \ , \tag{1.11}$$

where the mapping $R_V : V \to V^*$ is called the **Riesz map**. The **Riesz representation theorem** provides that R_V^{-1} exists and that $\|v\|_V = \|R_V(v)\|_{V^*} \ \forall v \in V$, where we are applying the concept of induced norms. Because the Riesz map gives us a one-to-one identification of elements in V and V^*, it is sometimes loosely said of Hilbert spaces that "$V = V^*$", and Riesz maps are often omitted as an abuse of notation, blurring the distinction between duality pairing and inner products.

Remark 3. Readers with a physics background are likely already familiar with this concept in a different notation. In bra–ket notation, the "bra" $\langle \psi |$ corresponding to the "ket" $| \psi \rangle$ is its Riesz representation.

Remark 4. Readers familiar with differential geometry may recognize the raising and lowering of indices to translate between vectors and covectors as an instance of Riesz representation, where covectors are in the dual of a manifold's tangent space and index lowering is the Riesz map.

If $A : U \to V$ maps between Hilbert spaces, we may extend the concept of the transpose to define its **adjoint**, $A^* : V \to U$ such that

$$(A(u), v)_V = (u, A^*(v))_U \quad \forall u \in U, \forall v \in V \ . \tag{1.12}$$

In terms of the Riesz maps for U and V, the adjoint and transpose are related by

$$R_U \circ A^* \circ R_V^{-1} = A^T \ , \tag{1.13}$$

which is often simplified, through abuses of notation, to "$A^* = A^T$".

An important type of linear operator on a Hilbert space is **orthogonal projection**. An orthogonal projector $P : U \to V$ mapping from a Hilbert space U to a subspace $V \subset U$ (where V inherits the inner product associated with U) has the property that

$$(u - Pu, v)_U = 0 \quad \forall v \in V, u \in U \ , \tag{1.14}$$

i.e., the difference between a vector u and its projection Pu onto V is orthogonal to every vector in V. An important property of P is that it is **idempotent**, i.e., $P^2 := P \circ P = P$:

$$(Pu - P^2 u, v)_U = 0 \quad \forall v \in V \quad \Rightarrow \quad Pu - P^2 u = 0 \quad \Rightarrow \quad P = P^2 \ , \tag{1.15}$$

because both Pu and $P^2 u$ are in V. Idempotent linear operators are generally referred to as (possibly non-orthogonal) **projectors**.

1.5 Generalized calculus

To apply the concepts of functional analysis to PDEs, we will need to introduce modern definitions of calculus operations, although we do so here in an abbreviated and non-rigorous manner.

1.5.1 Lebesgue integration

We shall do the greatest disservice to integrals, by more-or-less skipping over the concept of **Lebesgue integration**, which requires a thorough course on real analysis to cover in a precise way. For now, we simply note that, unless otherwise specified, all integrals shall be understood in this mysterious sense, whose main operational implication is that integrands differing on subsets of the domain with zero **measure** (i.e., length in 1D, area in 2D, etc.) are equivalent, and integrals over such sets are zero. A related jargon is that a condition holding on all points of a domain except possibly a set of measure zero is said to hold **almost everywhere**, frequently abbreviated to "**a.e.**" in formulas.

Remark 5. In a more thorough proof-based presentation, the main implication of Lebesgue integration would be that it allows for limits to commute with integration under certain conditions, although we shall not need to invoke this property directly. (However, certain facts stated here without proof rely on it.)

1.5.2 Distributions and weak derivatives

From classical calculus, we understand how to take (partial) derivatives of functions $f : \Omega \subset \mathbb{R}^n \to \mathbb{F}$ which are sufficiently smooth. However, there are a variety of reasons why we may wish to extend the idea of differentiation to functions which are not classically differentiable, either to allow us to choose more convenient constructions of function spaces to approximate PDE solutions or to accommodate non-smooth physical phenomena, like shock waves or point loads, which may occur in systems otherwise governed by classical PDEs.

We begin by going in the opposite direction, and assuming we only know how to perform classical differentiation on functions which have infinitely-many bounded derivatives. To avoid questions of what happens at boundaries or at infinity, we further restrict to **compactly supported** functions, which are nonzero only on a subset of the domain that does not intersect the boundary and whose closure is bounded. The set of infinitely-differentiable, compactly supported functions is denoted C_0^∞, or, for convenience, $\mathcal{D}$.

The dual of this space, $\mathcal{D}^*$, is referred to as the set of **distributions**.[3] A distribution $\tilde{f} \in \mathcal{D}^*$ is called a **regular distribution** if there exists a function f (or, more precisely, an equivalence class of functions matching a.e.) meeting technical conditions for the following Lebesgue integral to be well-defined:

$$\langle \tilde{f}, \varphi \rangle = \int_\Omega f(x)\varphi(x)\,dx \quad \forall \varphi \in \mathcal{D}\,. \tag{1.16}$$

A common abuse of notation we shall adopt here is to use the same symbol for $\tilde{f}$ and f, and ignore altogether that f is in fact an equivalence class of functions. Distributions that cannot be

[3]The astute reader will ask: What is the topology of $\mathcal{D}$ with respect to which elements of $\mathcal{D}^*$ are continuous? The answer is surprisingly-technical, and not even a metric topology, but these details are not useful for our purposes.

represented in this way are called **singular distributions**. An example of a singular distribution would be the Dirac delta at x, δ_x, which is defined by $\langle \delta_x, \varphi \rangle = \varphi(x)$, although, by analogy to regular distributions, the duality pairing is often written as an integral, in an abuse of notation. Because distributions (especially singular ones) are defined through their action on elements of $\mathcal{D}$, their properties only become apparent when "testing" them via duality pairing, and thus elements of $\mathcal{D}$ are referred to as **test functions** in this context.

Now suppose that f in (1.16) were smooth. Then integration by parts would give us

$$\int_\Omega \partial^\alpha f \varphi \, dx = (-1)^{|\alpha|} \int_\Omega f \partial^\alpha \varphi \, dx \,, \tag{1.17}$$

where the boundary term disappears due to the compact support of $\varphi \in \mathcal{D}$. The superscript α here is a **multi-index**, or, a tuple of non-negative integers indicating the numbers of partial derivatives in each spatial direction, and $|\alpha|$ is the sum of these integers, or, the overall order of the corresponding differential operator.[4] Motivated by the above, we define the **weak derivative** (or **distributional derivative**) $D^\alpha f \in \mathcal{D}^*$ of a distribution $f \in \mathcal{D}^*$ by

$$\langle D^\alpha f, \varphi \rangle := (-1)^{|\alpha|} \langle f, \partial^\alpha \varphi \rangle \quad \forall \varphi \in \mathcal{D} \,. \tag{1.18}$$

For f a sufficiently regular distribution, its weak derivative coincides with the classical partial derivative of its corresponding function. However weak differentiation applies more generally to arbitrary elements of $\mathcal{D}^*$, such as the Dirac delta. Thus, without loss of generality, we may interpret all spatial derivatives encountered as weak derivatives.

1.5.3 Differentiation with respect to functions

A further extension of the idea of differentiation that we will need when considering energy principles and solution methods for nonlinear PDE systems is the **Gateaux derivative**. This is simply the dimension-agnostic generalization of the directional derivative from multivariate calculus. In particular, if some (possibly nonlinear) function $f : U \to V$ maps between two Banach spaces, its Gateaux derivative at $x \in U$ in the direction $h \in U$ is given by

$$D_h f(x) = \frac{d}{d\epsilon} f(x + \epsilon h) \bigg|_{\epsilon=0} \,, \tag{1.19}$$

assuming f is sufficiently regular around x for the limit associated with the derivative with respect to ϵ to exist. In cases relevant to numerical PDEs, the space U will typically be a set of functions, and the Gateaux derivative thus differentiates a functional with respect to a function.

Remark 6. When U is a function space and V is a scalar field, one often sees the alternative notations

$$D_{\delta g} f = \delta f[g] = \int \frac{\delta f}{\delta g} \delta g \, dx \,. \tag{1.20}$$

We favor the first, here, for its direct correspondence with the `derivative` function in the FEniCS Unified Form Language (cf. Chapter 3). However, the second notation is more common in the applied literature. It is easy to see that the use of "δg" for the direction is consistent with $D_{\delta g} g = \delta g$.

[4] A more thorough explanation of this notation is provided in Section B.4 of the appendices.

The existence of a function "$\delta f/\delta g$" in the third notation is subject to various technical conditions on the regularity of f and the measure-theoretic interpretation of "$\int dx$". Further, this notation may lead to confusion in cases where $f(g)$ includes spatial derivatives of g and functions in U do not vanish at the boundary of a domain. We therefore avoid using it here.

Remark 7. Much of the theory for nonlinear analysis (e.g., proving the convergence of Newton iteration in function spaces) relies on the closely-related **Fréchet derivative**, which is subject to more stringent regularity conditions on f. However, where the Fréchet derivative exists, it coincides with the Gateaux derivative, and the definition of the Gateaux derivative provides a more practical formula for the derivations needed to formulate numerical methods.

1.6 Sobolev spaces

As mentioned in Section 1.4.1, the choice of what norm to use in an infinite dimensional linear space—such as a function space—is not trivial, since norms may not be equivalent, and elements that are close in one norm may be far apart in another. Because a norm of a function must condense information from its entire domain into a single number, it is natural to consider formulas involving integrals. Further, if we want to use the resulting topology to characterize solutions of PDEs, one might expect norms to include derivative information. These considerations motivate the introduction of **Sobolev norms**, which consist of Lebesgue integrals of weak derivatives of functions. In particular, we define the family of seminorms

$$|f|_{W^{m,p}(\Omega)} = \left\{ \sum_{|\alpha|=m} \int_{\Omega} |D^{\alpha} f|^{p} \ dx \right\}^{1/p} \tag{1.21}$$

which may be used construct the norms

$$||f||_{W^{m,p}(\Omega)} = \left\{ \sum_{n \leq m} |f|^{p}_{W^{n,p}(\Omega)} \right\}^{1/p} \tag{1.22}$$

for function $f : \Omega \to \mathbb{F}$ and non-negative integers m and $p \geq 1$. The **Sobolev spaces** $W^{m,p}(\Omega)$ are the corresponding sets of functions for which the norms $|| \cdot ||_{W^{m,p}(\Omega)}$ are well-defined.

Some special cases of Sobolev spaces receive their own notations in the literature. The **Lebesgue spaces** are defined as

$$L^{p}(\Omega) := W^{0,p}(\Omega) \tag{1.23}$$

and the **Hilbert–Sobolev spaces** (often informally called "H-spaces") are defined by

$$H^{m}(\Omega) := W^{m,2}(\Omega) \, . \tag{1.24}$$

The space $L^2 = H^0$ is both a Lebesgue space and a Hilbert–Sobolev space. The H-spaces are obviously **nested**, in the sense that $H^m \subset H^n$ for $n < m$. As their name implies, H^m are Hilbert spaces, in the sense defined by Section 1.4.3. The corresponding inner product is

$$(u, v)_{H^{m}(\Omega)} = \sum_{|\alpha| \leq m} \int_{\Omega} D^{\alpha} u \, D^{\alpha} \overline{v} \, dx \, . \tag{1.25}$$

The duals of H-spaces are typically denoted by $H^{-m} := (H^m)^*$, which allows for a formal unification of some formulas relating the regularities of function(al)s appearing in PDE and approximation problems.

Remark 8. The space L^2 is likely already familiar to many practitioners as the set of "square-integrable functions" or, in some introductory physics texts, "Hilbert space" (with great loss of generality).

Remark 9. In a more canonical presentation of functional analysis, the Lebesgue spaces would be defined early-on and studied as examples of Banach spaces (or even assumed to be familiar from prerequisite coursework), and Sobolev spaces would be defined later in terms of them. The reverse approach is taken here for brevity.

Remark 10. What is the interpretation of L^p as $p \to \infty$? The L^∞ norm of a function f is defined as

$$\|f\|_{L^\infty(\Omega)} := \operatorname*{ess\,sup}_{x\in\Omega}|f(x)| := \inf_{z\in A}\left(\sup_{x\in\Omega\setminus z}|f(x)|\right) , \tag{1.26}$$

where A is the set of subsets of Ω with measure zero and "ess sup" is read "essential supremum". Qualitatively, this is selecting the supremum of $|f|$ while disregarding outlier values on sets of measure zero. It can be shown that, for $f \in L^\infty$, $\lim_{p\to\infty} \|f\|_{L^p} = \|f\|_{L^\infty}$. The interpretation of this is that increasing p places more emphasis on peak values. $W^{m,\infty}$ is defined analogously.

Remark 11. One sometimes needs to treat a particular coordinate of a function's domain separately from the others. In PDEs modeling physical systems, this coordinate is usually time. The idea of a **time-dependent Sobolev space** is to replace a space–time function $f : \Omega \times \mathcal{I} \to \mathbb{R}$ with a mapping $\mathbf{f} : \mathcal{I} \to W^{m,p}(\Omega)$ from the time interval $\mathcal{I} \subset \mathbb{R}$ to a Sobolev space on the spatial domain $\Omega \subset \mathbb{R}^d$. Then, an auxiliary function $\tilde{f} : \mathcal{I} \to \mathbb{R}$ can be defined such that $\tilde{f}(t) := \|\mathbf{f}(t)\|_{W^{m,p}(\Omega)}$, leading to a space–time norm

$$\|f\|_{W^{n,q}(\mathcal{I},W^{m,p}(\Omega))} := \left\|\tilde{f}\right\|_{W^{n,q}(\mathcal{I})} , \tag{1.27}$$

which defines a corresponding time-dependent Sobolev space $W^{n,q}\left(\mathcal{I}, W^{m,p}(\Omega)\right)$ with different regularities in space and time, where $W^{n,q}/W^{m,p}$ may instead be denoted L^q/L^p or H^n/H^m if applicable. In practice, though, this notation is often viewed as burdensome, and many references only indicate the spatial regularity of time-dependent functions.

Historically, the H-spaces were defined through a different construction involving the Fourier transform, and some references consider that to be the definition of these spaces. However, it results in an equivalent topology, and the above presentation is more useful for the purposes of understanding properties of finite element methods. One of the few concepts that is perhaps clearer in the Fourier construction is the extension to **fractional-order Sobolev spaces**, e.g., $H^{1/2}$. We shall not dwell on the precise interpretation of this, aside from noting that the smoothness of fractional-order spaces is, heuristically, in between that of adjacent integer-order spaces. The Heaviside function on the interval $\Omega = (-1, 1)$ is a practically-significant example of a function with fractional regularity; while obviously in $L^2(\Omega)$, having only a jump discontinuity while being otherwise smooth actually makes it somewhat more regular, and it is in fact in $H^{1/2-\epsilon}(\Omega)$, $\forall\epsilon > 0$.

One place where the technicalities of Lebesgue integration come to the fore is at boundaries. Because the boundary $\partial\Omega$ of Ω has zero measure, Sobolev functions do not have well-defined restrictions to it. To get around this, we introduce the concept of a **trace operator**, tr_Γ, where Γ is a surface of one dimension lower than the domain. When applied to a smooth function f, it corresponds to restriction: $\mathrm{tr}_\Gamma f = f|_\Gamma$. However, invoking the topological concept of density, and relying on the density of smooth functions in Sobolev spaces (which we shall not prove here), we can define $\mathrm{tr}_\Gamma : H^s(\Omega) \to H^{s-1/2}(\Gamma)$ (for $s > 1/2$) as the extension of the restriction map to the full Sobolev space, under some technical conditions on the geometry of Γ, and, again, glossing over the precise meaning of the fractional-order range space.

A case of particular interest to the study of PDEs and numerical methods is the subset of $H^m(\Omega)$ with with traces of derivatives up to order $m-1$ (including zeroth-order) vanishing on the boundary $\partial\Omega$. This subset is denoted $H^m_0(\Omega)$, and defined via

$$H^m_0(\Omega) := \overline{\mathcal{D}(\Omega)}\,, \tag{1.28}$$

where the topological closure is taken in $H^m(\Omega)$. The space $H^1_0(\Omega)$ is particularly prevalent in presentations of the finite element method focused on second-order elliptic problems, and it is sometimes non-rigorously defined as "$\{u \in H^1(\Omega) \text{ s.t. } u = 0 \text{ on } \partial\Omega\}$", although, as discussed above, the function "on the boundary" is not well-defined, and must be understood in the sense of a trace.

Two useful results about Sobolev spaces which are invoked frequently in the analysis of finite element methods are

- The **trace theorem**: $\exists C > 0$ (which depends on the domain geometry) such that

$$\|\mathrm{tr}_{\partial\Omega} u\|_{L^2(\partial\Omega)} \le C\|u\|_{H^1(\Omega)} \quad \forall u \in H^1(\Omega)\,, \tag{1.29}$$

  subject to some technical conditions on the shape of Ω.

- The **Poincaré inequality**: $\exists C > 0$ (again depending on domain geometry) such that

$$\|u\|_{L^2(\Omega)} \le C|u|_{H^1(\Omega)} \quad \forall u \in H^1_0(\Omega)\,, \tag{1.30}$$

  again subject to technical conditions on Ω.

Remark 12. The Poincaré inequality implies that the $H^1(\Omega)$ seminorm is in fact a norm on $H^1_0(\Omega)$.

Remark 13. The "technical conditions" on domain geometry referenced here depend on the concept of a **Lipschitz domain**, which we will not define precisely. Qualitatively, a domain's degree of Lipschitz regularity is determined by the regularity of a collection of mappings that flatten overlapping portions of its boundary. Examples of non-Lipschitz domains include regions with cusp-shaped boundaries, although most domains arising in practical examples are Lipschitz.

1.7 Linear elliptic PDEs

We are finally equipped with the mathematical tools needed to discuss the problem class of interest: PDEs. In particular, we consider linear elliptic problems, which most readily elucidate the role of functional analysis in understanding (numerical approximation of) PDEs.

1.7.1 The Poisson problem

We illustrate the main ideas with the numerical analyst's favorite model problem, namely the Poisson equation, which, in **strong form**, is: Given $f : \Omega \to \mathbb{R}$, find $u : \overline{\Omega} \to \mathbb{R}$ such that

$$-\Delta u(x) = f(x) \quad \forall x \in \Omega\,, \tag{1.31}$$
$$u(x) = 0 \quad \forall x \in \partial\Omega\,, \tag{1.32}$$

where $\Delta = \mathrm{div} \circ \mathrm{grad}$ is the Laplace operator.[5] Interpreting (1.31) in the sense of distributions and using the definition of a weak derivative,

$$\langle -\Delta u, v\rangle = \langle \nabla u, \nabla v\rangle \quad \forall v \in \mathcal{D}(\Omega)\,. \tag{1.33}$$

If we seek $u \in H^1_0(\Omega)$, then ∇u is a regular distribution and

$$\langle \nabla u, \nabla v\rangle = (\nabla u, \nabla v)_{L^2(\Omega)}\,. \tag{1.34}$$

The right-hand side of (1.34) remains well-defined for $v \in H^1_0(\Omega)$, which motivates us to consider the **weak problem**: Find $u \in H^1_0(\Omega)$ such that $\forall v \in H^1_0(\Omega)$,

$$a(u, v) = L(v)\,, \tag{1.35}$$

where

$$a(u, v) = (\nabla u, \nabla v)_{L^2(\Omega)} \tag{1.36}$$

and

$$L(v) = \langle f, v\rangle\,, \tag{1.37}$$

and we assume $f \in H^{-1}(\Omega)$. Because $a(\cdot,\cdot)$ is an inner product on $H^1_0(\Omega)$, we immediately have that such a u exists, due to the Riesz representation theorem:

$$u = R^{-1}_{H^1_0(\Omega)}(f)\,, \tag{1.38}$$

where the Riesz map corresponds to the inner product $a(\cdot,\cdot)$. Riesz representation also bounds u in $H^1_0(\Omega)$, in terms of the induced norm of the given problem data $f \in H^{-1}$. This illustrates how weakening the problem by considering less smooth function spaces allowed us to easily show that a solution exists, and bound its norm.

Remark 14. Enforcing a boundary condition in a weak problem through the choice of function space, as done here, is known as **strong enforcement**. See Exercise 10 for an example of weak enforcement.

Because (1.35) holds $\forall v \in H^1_0(\Omega)$, then it must also hold $\forall v \in \mathcal{D}(\Omega) \subset H^1_0(\Omega)$, so u also satisfies the original distributional interpretation of the strong form. By making additional assumptions about the regularity of f and shape of Ω, it is possible to show that u is also more regular than H^1, and, under certain conditions, satisfies the strong problem in the classical pointwise sense, although we shall not go into great detail about PDE regularity theory.

[5]The Laplace operator is often written "∇^2" in the applied literature, but this notation carries some risk of being interpreted as $\mathrm{grad} \circ \mathrm{grad}$.

1.7.2 The Lax–Milgram theorem

Generalizing the presentation of Section 1.7.1, consider an abstract problem of the form: Find $u \in V$ such that for all $v \in V$,

$$a(u, v) = L(v)\,, \tag{1.39}$$

where V is Hilbert, a is bilinear, and L is linear. It is an obvious extension of the discussion from Section 1.7.1 that one can show existence of a bounded u if $L \in V^*$ and $a(\cdot,\cdot)$ is a valid inner product on V. However, these conditions can be weakened to

- a is **bounded**: $\exists C_\mathrm{B} \geq 0$ such that $a(u, v) \leq C_\mathrm{B}\|u\|\,\|v\| \quad \forall u, v \in V$.
- a is **coercive**: $\exists C_\mathrm{C} > 0$ such that $a(v, v) \geq C_\mathrm{C}\|v\|^2 \quad \forall v \in V$.

That these conditions imply the existence of a solution u bounded by the induced norm of L is known as the **Lax–Milgram theorem**. Notice that symmetry is *not* required.

1.7.3 The inf-sup condition

While sufficient for many practical situations, even the hypotheses of the Lax–Milgram theorem are unnecessarily strong. More generally, we may consider the problem: Find $u \in U$ such that for all $v \in V$,

$$a(u, v) = L(v)\,, \tag{1.40}$$

where U and V are Hilbert.[6] Then coercivity from the Lax–Milgram theorem can be weakened yet further by replacing it with the conditions

- The **inf-sup condition**: $\exists C_\mathrm{S} > 0$ such that

$$\inf_{\substack{u \in U \\ \|u\|_U = 1}} \sup_{\substack{v \in V \\ \|v\|_V = 1}} a(u, v) \geq C_\mathrm{S}\,. \tag{1.41}$$

- Non-degeneracy: $\sup_{u \in U} a(u, v) > 0 \quad \forall v \in V, v \neq 0$.

These conditions are obviously satisfied by any coercive form a, but also by many non-coercive forms, some examples of which will be discussed in later chapters.

Remark 15. Some alternate names for this result are: the Babuška–Brezzi condition, the Ladyzhenskaya–Babuška–Brezzi (LBB) condition, the generalized Lax–Milgram theorem, and the Babuška–Lax–Milgram theorem.

[6]Actually, this condition on the spaces is unnecessarily-strong. The result still holds if U and V are Banach and V is **reflexive**, but the concept of reflexivity was omitted to streamline the present exposition. In examples relevant to finite elements, the spaces will always be Hilbert.

1.8 Further reading

As noted in Section 1.1, the coverage of this chapter is intended to be as minimal as possible for following the rest of the chapters. Readers curious about the mathematical foundations of functional analysis and PDE theory might consult some of the following references:

- The standard undergraduate introductory text on mathematical analysis is a concise book by Rudin [6] (sometimes affectionately called "baby Rudin", to distinguish it from a much longer text on functional analysis by the same author).
- The text of Oden and Demkowicz [7] covers real and functional analysis rigorously, assuming the reader begins with an undergraduate engineering background, and emphasizing topics important to numerical analysis of PDEs. The explanation of Lebesgue integration is particularly good.
- Kreyszig [8] is a standard graduate-level text on functional analysis.
- The draft text by Arbogast and Bona [9] covers functional analysis at a graduate level, giving priority to topics relevant to numerical analysis of PDEs, and providing many challenging exercises.
- The monograph by Adams [10] is a comprehensive reference on Sobolev spaces.
- The recent lecture notes of Demkowicz [11] provide a good overview of Sobolev spaces (referred to therein as "energy spaces").
- Evans [5] is a standard graduate-level text on PDE theory.

However, these references should not be necessary to follow subsequent chapters.

1.9 Exercises

1. Show that (1.5) satisfies the definition of a metric.
2. Show that (1.8) satisfies the definition of a norm.
3. Prove (1.10).
4. Show that the distributional derivative of the Heaviside function is a Dirac delta. Further, what is the derivative of the Dirac delta?
5. Construct a sequence of functions that converges to zero in $L^2((0,1))$ but diverges in $H^1((0,1))$. *Hint:* An oscillatory function with a small amplitude but high frequency can simultaneously have a small L^2 norm and a large H^1 norm.
6. Adapt the argument of Section 1.7.1 to the case where u has nonzero boundary conditions, i.e., $\operatorname{tr}_{\partial\Omega} u = g \not\equiv 0$. Assume that $g = \operatorname{tr}_{\partial\Omega} \ell_g$, where $\ell_g \in H^1(\Omega)$ is known. *Comment:* ℓ_g is commonly referred to as a **lift** of g, by the **lift operator** $\ell_{(\cdot)}$. *Hint:* Reformulate the problem to solve for $\tilde{u} := u - \ell_g \in H^1_0(\Omega)$.

7. Suppose that bilinear form $a : V \times V \to \mathbb{R}$ and linear form $L : V \to \mathbb{R}$ satisfy the hypotheses of the Lax–Milgram theorem, and, additionally, a is symmetric, i.e., $a(u,v) = a(v,u)\ \forall u, v \in V$. Show that the solution $u \in V$ to $a(u,v) = L(v)\ \forall v \in V$ minimizes the functional

$$J(u) := \frac{1}{2}a(u,u) - L(u)\,. \tag{1.42}$$

Hint: Show that $J(v) \geq J(u)\ \forall v \in V$. Note that, because V is closed under linear combination, an arbitrary element $v \in V$ can be re-written as $u + w$, where $w := v - u \in V$.

8. Show that the weak form of the Poisson problem can be obtained by setting to zero the Gateaux derivative of $J(u) = \frac{1}{2}|u|^2_{H^1(\Omega)} - (f,u)_{L^2(\Omega)}$ in the direction of some test function v.

9. Consider the strong form of the advection–diffusion problem: Given $\kappa > 0$, $f : \Omega \to \mathbb{R}$, and $\mathbf{a} : \Omega \to \mathbb{R}^n$ with $\nabla \cdot \mathbf{a} = 0$, find $u : \overline{\Omega} \to \mathbb{R}$ such that

$$-\kappa\Delta u(x) + \mathbf{a}(x)\cdot\nabla u(x) = f(x) \quad \forall x \in \Omega\,, \tag{1.43}$$
$$u(x) = 0 \quad \forall x \in \partial\Omega\,. \tag{1.44}$$

Derive a weak form posed over $H^1_0(\Omega)$, and show that it satisfies the hypotheses of the Lax–Milgram theorem. *Hint:* Consider integrating half of the **a** term by parts.

10. Consider the following weak problem: Find $u \in H^1(\Omega)$ such that $\forall v \in H^1(\Omega)$

$$(\nabla u, \nabla v)_{L^2(\Omega)} + (u,v)_{L^2(\Omega)} = (f,v)_{L^2(\Omega)} + (h,v)_{L^2(\partial\Omega)}\,, \tag{1.45}$$

where $f \in L^2(\Omega)$ and $h \in L^2(\partial\Omega)$ are given data.

(a) What is the corresponding strong problem?

Hint: Integration by parts will involve a boundary term for problems posed on H^1 (as opposed to H^1_0). In particular, recall the vector calculus identity (A.6).

Comment: This is an example of **weak enforcement** of a boundary condition.

(b) Show that this satisfies the hypotheses of the Lax–Milgram theorem. (Coercivity and boundedness of a are easy here, but: Why is the boundary term of L in V^*? *Hint:* Use the trace theorem.)

11. Consider the un-forced unsteady heat equation: Find $u : \overline{\Omega} \times \mathbb{R}^+ \to \mathbb{R}$ such that

$$\partial_t u(x,t) - \Delta u(x,t) = 0 \quad \forall (x,t) \in \Omega \times \mathbb{R}^+\,, \tag{1.46}$$
$$u(x,t) = 0 \quad \forall (x,t) \in \partial\Omega \times \mathbb{R}^+\,, \tag{1.47}$$
$$u(x,0) = u_0(x) \quad \forall x \in \Omega\,, \tag{1.48}$$

where u_0 is given. Show how coercivity in the $H^1_0(\Omega)$ weak form of the corresponding steady problem corresponds to stability (i.e., decay over time) of the quantity $\frac{1}{2}\|u\|^2_{L^2(\Omega)}$. *Hint:* Multiply (1.46) by u and integrate over Ω.

Chapter 2

Survey of finite element analysis

2.1 Introduction

Just like the mathematical subjects of functional analysis and PDEs, the finite element method (FEM) deserves a course sequence of its own. Readers interested in the subject are strongly encouraged to take a course dedicated to the implementation details of FEA (following, e.g., [4]), although such coursework is not a strict prerequisite for the present material. The focus of this chapter is to state the method in an abstract way using the language of functional analysis, expose its key properties, and survey some common variants. The translation of abstract variational problem statements into code can be carried out systematically enough to be automated, which is the goal of software projects like FEniCS (as discussed at length in Chapter 3), even if the present implementations do not yet cover all cases of practical interest. It is nonetheless instructive (and imparts much practical knowledge) to "look under the hood" at the algorithms and data structures, but that is the purview of a more narrowly-scoped presentation.

Another point to address at the outset is the question of when to use FEM as opposed to other methods for numerical PDEs, such as finite differences or finite volumes. This is an especially relevant question to the topic of solving coupled problems, as different application domains may traditionally use different numerical methods. However, we show in Section 2.5 that the systems of algebraic equations resulting from finite difference and finite volume methods can actually be obtained through the FEM formalism. Taking this unified viewpoint is of value when coupling discretizations of different problems, since all of the interacting subproblems will "speak the same language". However, the prevailing pedagogical philosophy in engineering—especially fluid dynamics—is that the variational machinery needed for this unification distracts from other important concepts in numerical PDEs (like stability, consistency, conservation, and efficient algebraic solvers), which can be communicated with less mathematical abstraction in the finite difference or volume settings. To clarify foundational concepts in numerical analysis, it is strongly recommended to take introductory finite difference-based coursework on numerical ordinary and partial differential equations.

2.2 The (Bubnov–)Galerkin method

In Section 1.7.1, we worked through an example of how to convert PDE systems into weak problems of the form: Find $u \in V$ such that $\forall v \in V$

$$a(u, v) = L(v) , \tag{2.1}$$

where a and L are forms, and V is a linear space. To understand coupled problems, we want to develop the theory of finite element methods in a *generic* way that can be applied to multiple different PDE systems, describing different physical phenomena, so we proceed with this abstract problem statement. Assume L is linear and a is linear in its second argument, and consider the following related problem: Find $u^h \in V^h$ such that $\forall v^h \in V^h$

$$a\left(u^h, v^h\right) = L\left(v^h\right) , \tag{2.2}$$

where V^h is a N-dimensional subspace of V. By definition, then, there is some basis $\{\phi_i\}_{i=1}^N$ such that

$$V^h = \text{span}\{\phi_1, \cdots, \phi_N\} . \tag{2.3}$$

Using the semilinearity of a, we therefore obtain a system of N equations,

$$\left\{a(u^h, \phi_i) = L(\phi_i)\right\}_{i=1}^N , \tag{2.4}$$

for N unknowns, namely the coordinates of u^h in the basis $\{\phi_i\}$:

$$u^h = u_1\phi_1 + \cdots + u_N\phi_N . \tag{2.5}$$

This method of reducing an infinite-dimensional variational problem to a finite algebraic system of equations is called **Galerkin's method**, or, to distinguish it from a variety of related methods, the particular approach shown here—namely posing the original problem unaltered on a subspace of V—is sometimes referred to as the **Bubnov–Galerkin method**.

Solution of large nonlinear algebraic systems is fraught with both theoretical and practical difficulties, but, if a is also linear in its first argument, then it can easily be seen that (2.4) is a **linear system** of equations, of the form

$$\mathbf{Au} = \mathbf{b} , \tag{2.6}$$

where

$$A_{ij} = a(\phi_j, \phi_i) \quad \text{and} \quad b_i = L(\phi_i) \quad \forall i, j \in \{1, \cdots, N\} . \tag{2.7}$$

If the finite-dimensional problem (2.2) satisfies the hypotheses of the inf-sup theorem, then $\mathbf{A}^{-1}$ exists, and we can obtain $\mathbf{u}$ from a linear solve.

Remark 16. Although the examples of PDEs we have packaged in the form (2.1) have had a single scalar field, our discussion of Galerkin's method here is not tied to any particular structure of V beyond it being a Hilbert space. For a **coupled problem** involving N unknown fields, in Hilbert spaces $V_1, \ldots, V_N$, we can define the **mixed function space** $V := V_1 \times \cdots \times V_N$ to obtain a problem of the same structure. One valid choice of inner product on V would be

$$(u, v)_V := \sum_{i=1}^N (u_i, v_i)_{V_i} , \tag{2.8}$$

where numerical subscripts select tuple components of elements of V. In physical problems where $\{V_i\}$ have different units, the terms of (2.8) might be weighted by dimensional coefficients.

Remark 17. Galerkin's method is not just a numerical tool for scientists and engineers to approximate solutions; it plays an important role in PDE theory as well, e.g., in the proof of solution existence for linear parabolic PDEs.

Remark 18. The classical analytical method of deriving orthogonal eigenfunctions of a partial differential operator (usually by separation of variables) and expressing the PDE solution as a linear combination of these is an example of Galerkin's method. In that case, the matrix to invert is diagonal, so there is an explicit formula for each coefficient.

Remark 19. If (2.4) is in fact nonlinear in u^h, then the standard approach is to seek a solution through **Newton iteration**. This uses the concept of the Gateaux derivative to **linearize** the problem. Given some current guess $u^h_i \in V^h$ for the solution, we define the **residual** form

$$R(u^h_i, v^h) := a(u^h_i, v^h) - L(v^h) \tag{2.9}$$

which is linear in v^h. Its Gateaux derivative with respect to the first argument (or its **Jacobian**)

$$J(u^h_i, v^h, \Delta u^h) := D_{\Delta u^h} R(u^h_i, v^h) \tag{2.10}$$

is linear in the new argument Δu^h_i. Then the problem: Given u^h_i, find Δu^h_i such that for all v^h

$$J(u^h_i, v^h, \Delta u^h_i) = -R(u^h_i, v^h) \tag{2.11}$$

results in a linear algebraic system, which can be solved[1] with Newton iteration:

1. Solve (2.11) for Δu^h_i.
2. $u^h_{i+1} := u^h_i + \Delta u^h_i$.
3. $i \leftarrow i + 1$ and return to Step 1.

until the **residual vector**

$$\Big(R(u^h_i, \phi_1)\,,\ \cdots\ ,\ R(u^h_i, \phi_N)\Big)^T \in \mathbb{R}^N \tag{2.12}$$

is small enough to meet a(n application-dependent) convergence criterion.

2.2.1 Error analysis for linear problems

The solution u^h to (2.2) is nominally an approximation of the solution u to (2.1). How do we know whether it is a good approximation? Let us assume that a and L satisfy the hypotheses of the inf-sup theorem for the continuous and discrete problems. In particular, that means $\exists C^h_{\mathrm{S}} > 0$ such that

$$\inf_{\substack{u^h\in V^h\\ \|u^h\|=1}} \sup_{\substack{v^h\in V^h\\ \|v^h\|=1}} a\left(u^h, v^h\right) \geq C^h_{\mathrm{S}}\,, \tag{2.13}$$

[1] The problem and initial guess actually need to satisfy very strict criteria for convergence of Newton iteration to be *provably* guaranteed. In practice, engineers ignore these criteria, try anyway, and augment the basic algorithm with features like **line searching** to improve robustness.

where $\|\cdot\|$ is some norm on V, and $\exists C_\mathrm{B}$ such that

$$a(u,v) \le C_\mathrm{B}\|u\|\,\|v\| \quad \forall u,v \in V\,. \tag{2.14}$$

Next, consider some arbitrary $w^h \in V^h$ and define the **total error**

$$e := u - u^h \in V\,, \tag{2.15}$$

the **interpolation error**

$$\eta := u - w^h \in V\,, \tag{2.16}$$

and the **method error**

$$e^h := w^h - u^h \in V^h\,. \tag{2.17}$$

Clearly we have the relation that $e = e^h + \eta$. Because $V^h \subset V$, we have the property of **strong consistency**, sometimes called **Galerkin orthogonality**, by analogy to the case where a satisfies the more restrictive properties of an inner product (cf. Remark 20):

$$a\left(u,v^h\right) = L\left(v^h\right) \quad \Rightarrow \quad a\left(e,v^h\right) = 0 \quad \forall v^h \in V^h\,. \tag{2.18}$$

This implies that

$$\frac{a\left(e,v^h\right)}{\|e^h\|\,\|v^h\|} = \frac{a\left(e^h+\eta,v^h\right)}{\|e^h\|\,\|v^h\|} \quad \Rightarrow \quad \frac{a\left(e^h,v^h\right)}{\|e^h\|\,\|v^h\|} = -\frac{a\left(\eta,v^h\right)}{\|e^h\|\,\|v^h\|} \qquad \forall v^h \ne 0\,, \tag{2.19}$$

where we disregard the case of zero method error as trivial. Combining (2.19) with the inf-sup condition,

$$C^h_\mathrm{S} \le \sup_{v^h} \frac{a\left(e^h,v^h\right)}{\|e^h\|\,\|v^h\|} = \sup_{v^h} \frac{a\left(\eta,-v^h\right)}{\|e^h\|\,\|v^h\|}\,, \tag{2.20}$$

where we omit obvious conditions on v^h for visual clarity. Now using boundedness of a, multiplying through by $\|e^h\|$, and dividing through by C^h_S, we control the method error in terms of the interpolation error:

$$\|e^h\| \le \frac{1}{C^h_\mathrm{S}} \sup_{v^h} \frac{C_\mathrm{B}\|\eta\|\,\|v^h\|}{\|v^h\|} = \frac{C_\mathrm{B}}{C^h_\mathrm{S}}\|\eta\|\,. \tag{2.21}$$

Using the triangle inequality,

$$\|e\| \le \|e^h\| + \|\eta\| \le \left(\frac{C_\mathrm{B}}{C^h_\mathrm{S}} + 1\right)\|\eta\|\,, \tag{2.22}$$

and the fact that w^h is arbitrary gives us

$$\|e\| \le \left(\frac{C_\mathrm{B}}{C^h_\mathrm{S}} + 1\right) \inf_{w^h \in V^h} \|u - w^h\|\,. \tag{2.23}$$

This result essentially shows us that the Galerkin solution's error is within some factor of the error in the *best possible* choice of approximation available in the discrete space. It reduces the problem of estimating the error of a numerical method to approximation theory. One can systematically construct sequences of finite-dimensional spaces, such as the finite element spaces introduced in Section 2.4, for which this approximation error converges to zero. Showing convergence of the approximation error to zero for a simple case is left for the reader as Exercise 2.

However, there is a big caveat: The inf-sup constant C_{S}^h must be uniformly bounded from below over the (tail of the) sequence of finite-dimensional spaces, which is not always trivial. In the special case of the Lax–Milgram theorem, we can simplify (2.13) to

$$\inf_{\substack{v\in V\\ \|v\|=1}} a\left(v,v\right) \geq C_{\text{S}}\ , \tag{2.24}$$

which is simply an alternate form of the coercivity condition. In (2.24), if we replaced V with $V^h \subset V$, then the left-hand side could only get bigger or stay the same; the infimum over a subset set cannot be smaller than the infimum over the full set. Thus, coercivity of the continuous problem automatically implies coercivity for any Galerkin discretization, with (at least) the same constant. The same is *not* true for problems that are only inf-sup stable. This is why (2.13) is written in terms of V^h, not V. The supremum may get smaller when restricting to a smaller space. For these problems, sequences of V^h must be chosen very carefully, so that (2.13) is satisfied with C_{S}^h independent of the level of refinement. Later chapters will return to this issue in the context of saddle point problems enforcing constraints, such as incompressibility (Chapter 5) or kinematic compatibility (Chapter 7).

Remark 20. If the bilinear form a is an inner product, (2.18) clearly implies that Galerkin's method is the orthogonal projection (as defined by (1.14)) of the exact solution onto V^h. This provides a clear geometric intuition for why u^h is the closest discrete function to u in the topology induced by a. (The rigorous proof that it is the unique closest approximation is a special case of Exercise 1 of this chapter, where $C_B = C_C = 1$.)

Remark 21. We can interpret the error analysis of this section in terms of the classical concepts of consistency and stability. The notions of consistency are:

- The method is consistent with the continuous problem, which manifests as the Galerkin orthogonality property.
- The finite element function space can be made to approximate the exact solution to within a desired tolerance by refining the mesh.

The notion of stability is:

- The numerical method's finite-dimensional variational problem satisfies the hypotheses of the Lax–Milgram or inf-sup theorem uniformly under mesh refinement, which implies that the method does not amplify the truncation error inherent to using a finite-dimensional function space.

The classical result that consistency and stability are necessary and sufficient conditions for convergence is the Lax equivalence theorem.

2.3 Beyond Galerkin

There are many reasons why one might not want to use the approach outlined and analyzed in Section 2.2. A few examples of such reasons are:

- a is not coercive, and there is no obvious way to choose a sequence of V^hs for which an inf-sup condition holds uniformly.
- The ratio of coercivity to boundedness constants for a is very small, so the error estimate (2.23) is not helpful.
- Constructing V^h such that it is contained in V may be inconvenient, or prevent one from ensuring that V^h satisfies other desirable properties.
- The algebraic equation system resulting from Galerkin's method may be difficult to solve efficiently.

This section will briefly touch on some other classes of methods, which, depending on context, are sometimes also referred to as "Galerkin methods", but are distinct from the Bubnov–Galerkin method of Section 2.2.

2.3.1 Petrov–Galerkin methods

The term "Petrov–Galerkin method" is widely-used in the FEA literature, although neither its precise definition nor even the identity of its co-namesake Petrov[2] seem to be universally agreed-upon. For our purposes, consider the definition (which is indeed at variance with certain uses in the literature) to be as follows. A **Petrov–Galerkin method** is one involving a variational problem derived from multiplying the strong PDE residual by some operator acting on the test function. (This obviously includes the Bubnov–Galerkin method as a special case, where that operator is the identity map). For example, if a strong PDE with solution u and data f is given by $\mathcal{L}u = f$, where $\mathcal{L}$ is a (distributional) partial differential operator, then the Bubnov–Galerkin method is: Find $u^h \in V^h$ such that for all $v^h \in V^h$

$$\left\langle \mathcal{L}u^h - f, v^h \right\rangle = 0 , \tag{2.25}$$

while a Petrov–Galerkin method would take the form

$$\left\langle \mathcal{L}u^h - f, \mathcal{L}^h_{\text{P–G}} v^h \right\rangle = 0 , \tag{2.26}$$

where the linear operator $\mathcal{L}^h_{\text{P–G}}$ is selected based on problem-specific criteria, and usually depends on V^h. This can be re-arranged into the form

$$a_h\left(u^h, v^h\right) = L_h\left(v^h\right) , \tag{2.27}$$

[2]The most plausible "Petrov" seems to be a Soviet scientist and contemporary of Boris Galerkin, Georgii I. Petrov, on the basis of a Russian-language paper from 1940. However, some references instead credit Bulgarian physicist Alexander G. Petrov.

where

$$a_h\left(u^h, v^h\right) := \left\langle \mathcal{L}u^h, \mathcal{L}^h_{\text{P–G}} v^h \right\rangle \quad \text{and} \quad L_h\left(v^h\right) := \left\langle f, \mathcal{L}^h_{\text{P–G}} v^h \right\rangle \tag{2.28}$$

differ from a and L. We will see later how this concept can be used to formally represent the coupling of the numerical solution u^h with unresolved fine-scale features of the exact solution u.

Remark 22. It is clear that, despite $a_h \neq a$ and $L_h \neq L$, Petrov–Galerkin methods (as defined here) are strongly-consistent.

2.3.2 Non-conforming methods

As mentioned above, one reason to depart from the Bubnov–Galerkin approach is that the assumption of $V^h \subset V$ may become inconvenient. Methods solving variational problems over $V^h \subset V$ are referred to as V-conforming, whereas methods in which $V^h \not\subset V$ are **non-conforming**. Usually extra effort is needed to formulate such methods in stable and consistent ways, and, like Petrov–Galerkin methods, the variational forms defining the discrete problem typically differ from those in the continuous problem (cf. (2.27)).

An example of the motivation for a non-conforming method might be to preserve the integral form of a conservation law associated to the strong PDE on some subset of the domain. The integral conservation law will hold on $\Omega_e \subset \Omega$ if V^h contains the characteristic function of Ω_e. However, the characteristic function is discontinuous, and would therefore not be available in a conforming discretization of a variational problem posed over $H^1(\Omega)$, which does not permit jump discontinuities across surfaces of dimension one less than Ω. This property of "local conservation" is often touted as a selling point of the so-called "discontinuous Galerkin" family of methods,[3] which will be discussed and compared with finite volume methods in Section 2.5.2.

2.3.3 Weakly-consistent methods

Another key assumption of the Bubnov–Galerkin method that is often violated by methods used in practice is strong consistency. For a **weakly-consistent method**, (2.18) does not hold, i.e.,

$$a_h\left(u, v^h\right) - L_h\left(v^h\right) \neq 0 \,, \tag{2.29}$$

but the consistency error diminishes as the function space V^h better approximates V, thereby allowing the method to converge. Based on the discussion so far, it may be difficult to imagine why one would deliberately add such a consistency error. However, it may, in many cases of practical interest, be *much* easier to formulate, implement, and/or solve a discrete problem that is only weakly consistent, while the resulting loss in accuracy may be minor relative to, e.g., the error inherent in modeling the physical system of interest by a PDE (which is often much larger than the numerical error e). One of the most prevalent types of weakly-consistent method seen in practice is the penalty regularization of constraints. As we will see later, in the context of incompressibility (Chapter 5) and fluid–structure kinematic constraints (Chapter 7), constrained problems may result in non-coercive variational forms, and all of the attendant inf-sup difficulties, whereas penalty approximations of constraints are coercive, and therefore easier to discretize and solve, at the cost

[3]This should not be misunderstood as the common misconception that continuous Galerkin methods are "not (locally) conservative". See [12] for the sense in which they are.

of being less accurate. As another example, we will see in Section 2.5.1 how the classical upwind finite difference scheme for advection is equivalent to a weakly-consistent finite element method that adds some H^1_0 coercivity to the Bubnov–Galerkin approach (although it can be corrected to become a strongly-consistent Petrov–Galerkin method without losing stability).

2.4 Finite element function spaces

In the foregoing discussion, we have not yet commented on what might be an appropriate choice of the finite basis $\{\phi_i\}$ for the approximation space V^h, beyond noting that we should have some knob to turn that increases its approximation power. For concreteness, consider first the Bubnov–Galerkin method of Section 2.2, applied to a linear problem. To efficiently compute the entries of the matrix $\mathbf{A}$ and vector $\mathbf{b}$ and solve the resulting linear system (2.6), one would ideally have a sparse $\mathbf{A}$, both to reduce the number of nonzero entries that need to be computed and to simplify the solution procedure. One way to achieve this that can be generalized to complex geometries is to construct a function space V^h in the following way, which is the defining feature of the **finite element method**.

First, split the PDE domain $\Omega \subset \mathbb{R}^d$ into a **mesh** of **elements** $\mathcal{T}_h = \{\Omega_e\}_{e=1}^{N_{\text{el}}}$, with disjoint interiors $\Omega^\circ_e \cap \Omega^\circ_f = \emptyset$ for $e \neq f$, and define a basis of **shape functions** (usually spanning polynomials up to some degree k) on $\mathbf{F}_e^{-1}(\Omega_e)$, where $\mathbf{F}_e : \square \to \Omega_e$ maps some standard shape $\square \subset \mathbb{R}^d$ (e.g., a unit square or triangle in 2D) to Ω_e. Frequently, the components of the mapping $\mathbf{F}_e$ are themselves linear combinations of the shape functions, which is referred to as **isoparametric mapping**. This notation is illustrated for a mesh of 2D triangular elements in Figure 2.1. Taking the polynomial shape functions on each element as building blocks, then different classes of finite element spaces on Ω can be formed by imposing different types of continuity constraints across the facets of elements and/or defining different pushforward mappings from $\square$ to Ω_e to obtain the physical-space shape functions defined on Ω_e. The set of linear combinations of physical-space shape functions constrained to meet continuity criteria forms the space V^h appearing in the presentation of the Bubnov–Galerkin method of Section 2.2. The **basis functions** $\{\phi_i\}$ for that space are of course non-unique, but are usually selected for maximum sparsity, i.e., $a(\phi_j, \phi_i)$ is zero for as many pairs (i, j) as possible.

In a comprehensive course on finite element methods, significant time would be devoted to the construction of specific bases of shape functions that facilitate the imposition of different continuity constraints between elements. However, we leave these details to FEA texts such as [4], and merely survey the important mathematical properties of the resulting function spaces.

2.4.1 Continuous Galerkin spaces

The most widely-known type of finite element spaces are the **continuous Galerkin (CG)** spaces. Scalar fields defined in CG spaces are required to have C^0 continuity across inter-element boundaries, and shape functions are pushed forward by composition with $\mathbf{F}_e^{-1}$, i.e., the physical-space shape function $N_a : \Omega^e \to \mathbb{R}$ corresponding to the parametric shape function $\widehat{N}_a : \square \to \mathbb{R}$ would be

$$N_a = \widehat{N}_a \circ \mathbf{F}_e^{-1} \ . \tag{2.30}$$

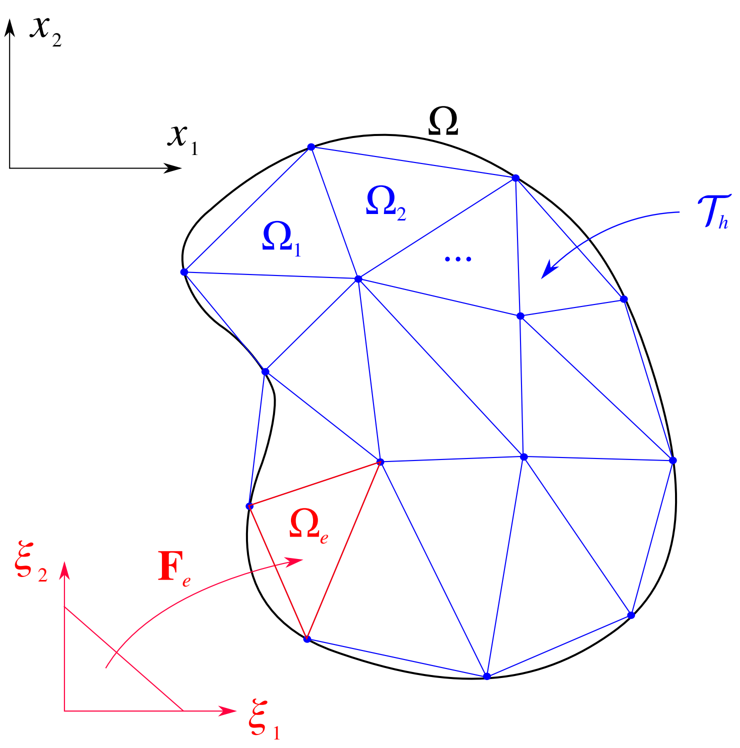

Figure 2.1: 2D illustration of notation used in the construction of finite element function spaces, for the case where $\Omega \subset \mathbb{R}^2$, $\square$ is a unit triangle, and the mappings $\{\mathbf{F}_e\}$ are affine.

Vector-valued (or higher-rank) CG function spaces simply have a scalar-valued CG space for each component.

The canonical basis for a CG space is constructed roughly as follows. Shape functions are defined as a **nodal basis**, such that there are several points in $\square$, called **nodes**, at which exactly one shape function is nonzero, with unit value. Then, the mappings $\{\mathbf{F}_e\}$ are selected such that images of nodes on shared boundaries between elements are coincident, and the nonzero shape functions at each unique nodal point are combined into a basis function, producing a nodal basis for the space V^h. This is illustrated for a 1D piecewise-linear CG space in Figure 2.2.

Based on the error estimate (2.23), recall that what we want to know about a given discrete space V^h is: What is the smallest distance in a given Sobolev norm between V^h and the exact solution u? Suppose that $\mathcal{T}_h$ is **quasi-uniform**, meaning that $\exists C > 0$ and $h > 0$ such that

$$C \max_e \{\operatorname{diam} \Omega_e\} \;\le\; h \;\le\; \frac{1}{C} \min_e \{\operatorname{diam} \Omega_e\}\ , \tag{2.31}$$

where h is the **global element size** or **refinement parameter**. If u is a scalar function that is in $H^r(\Omega)$ and V^h is a CG finite element space of polynomial degree k (which we denote as CG$_k$), then $\exists C > 0$ such that

$$\inf_{w^h \in V^h} \left\| w^h - u \right\|_{H^s(\Omega)} \le C h^\alpha \|u\|_{H^r(\Omega)}\ , \tag{2.32}$$

where $\alpha = \min\{k + 1 - s, r - s\}$ is the **convergence rate** (of interpolation error), and we tacitly make some reasonable technical assumptions on the regularity of the mappings $\{\mathbf{F}_e\}$, shape of Ω,

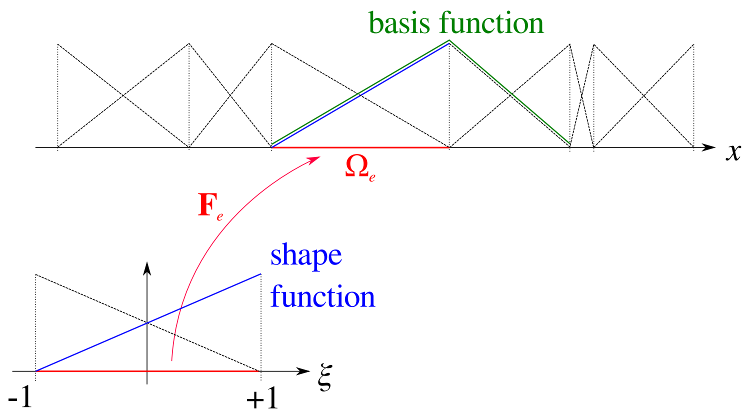

Figure 2.2: 1D illustration of the canonical basis for a piecewise-linear CG space, where $\Omega \subset \mathbb{R}$, $\square$ is a bi-unit interval, and the mappings $\{\mathbf{F}_e\}$ are affine. The nodes of this element are at $\xi = \pm 1$.

etc. In particular, if u is sufficiently regular, viz., $r > k + 1$, we have

$$\inf_{w^h \in V^h} \left\| w^h - u \right\|_{H^s(\Omega)} \leq C h^{k+1-s} \|u\|_{H^{k+1}(\Omega)} . \tag{2.33}$$

Proving some elementary cases of this estimate in 1D is left to the reader, as Exercise 2.

Another type of estimate that is useful in finite element spaces is an **inverse estimate**, which allows for bounding of higher Sobolev norms in terms of lower ones. Obviously, this is not possible in the continuous setting, because the lower Sobolev spaces are strictly larger than the higher ones. However, in a CG space, we have, on each element Ω_e, for all $r, s \geq 0$, $\exists C > 0$ such that for all $u^h \in V^h$,

$$\left\| u^h \right\|_{H^{s+r}(\Omega_e)} \leq C h^{-r} \left\| u^h \right\|_{H^s(\Omega_e)} , \tag{2.34}$$

where we essentially trade a power of h for each higher derivative bounded.

Remark 23. As we did for variational formulations in Remark 21, we can also relate the properties of function spaces to the classical concepts of consistency and stability. The interpolation estimate (2.32) expresses the consistency of V^h, while the inverse estimate (2.34) expresses its stability. The stabilizing effects of inverse estimates will be exploited in the analyses of Petrov–Galerkin formulations, in Chapters 5 and 6.

Remark 24. N.b. that the error analysis of Section 2.2 only ensures that total error e converges to zero at the **optimal rate** α of the interpolation estimate (2.32) if the norm $\|\cdot\|$ in which the method satisfies the inf-sup hypotheses is (equivalent to) $\|\cdot\|_{H^s(\Omega)}$. For some problems, it is possible to also show optimal convergence in weaker norms (e.g., of less-regular Sobolev spaces), but additional analysis is required to do this. (The most famous example of such analysis is referred to in the literature as the "**Aubin–Nitsche trick**", but it is only applicable to certain problems.)

Remark 25. Although one often comes across such statements in the literature, it is imprecise (at best) to say that a numerical method is "first-order", "second-order", vel sim., without specifying what norm the error is measured in. A method can simultaneously converge at different rates with respect to different norms.

Remark 26. When verifying the convergence of a finite element method in a problem with a known solution, one often computes the H^s norm of error in the numerical solution and compares its rate of convergence with the optimal rate α at which interpolation error converges. A convenient way to check this is to compute numerical solutions at several values of h, then plot the log of the H^s error as a function of $\log h$. The slope of the resulting curve in this log–log plot is the empirical convergence rate of the numerical method. For an optimal method, the slope should approach α as $h \to 0$.

2.4.2 Discontinuous Galerkin spaces

The **discontinuous Galerkin (DG)** spaces have no continuity constraints imposed between elements, and also use a pushforward of the form (2.30). This is convenient from the perspective of creating meshes, since continuity constraints place obvious restrictions on the arrangements of $\{\Omega_e\}$. However, these spaces are non-conforming for many common PDE systems, and require specially-designed discrete variational problems to be effective. DG methods also require more degrees of freedom than CG spaces of the same polynomial degree to approximate smooth functions with similar levels of error. However, the analogy between DG and finite volume methods (discussed further in Section 2.5.2) has made DG a popular approach for fluid mechanics in recent years.

2.4.3 Isogeometric function spaces

The term **isogeometric analysis (IGA)** refers to the use of Galerkin-like methods with spaces of spline functions similar to those used in computer-aided design and graphics. Viewed as an instance of the finite element framework described here, this essentially corresponds to imposing C^k continuity (for k up to $p - 1$) between elements. This imposes yet more constraints on the possible arrangements of elements, and the resulting smooth spline spaces are often used to express PDE solutions and/or geometry in homogeneous coordinate representations, but we shall not delve into the intricacies of constructing spline spaces. An important property of IGA function spaces is that the additional continuity allows application of straightforward conforming Bubnov–Galerkin methods for variational problems that, in the continuous setting, are posed over $V = H^k(\Omega)$ for $k > 1$. Just as the decreased continuity requirements of DG causes those spaces to require more degrees of freedom than CG for a given accuracy level, the increased continuity of IGA leads to smaller numbers of degrees of freedom.

2.4.4 Discrete de Rham complexes

Many PDEs relevant to modeling physical problems result from conservation laws, by relating derivatives integrated over a volume to fluxes integrated over the boundary of that volume. The various "integration-by-parts" formulas used to relate volumes and boundaries are unified by the **exterior calculus** formalism, in which an operator called the exterior derivative, $\boldsymbol{d}$, is dual to boundary restriction and acts on a class of tensor fields called **differential forms**. We will not attempt to cover exterior calculus, but a key property of $\boldsymbol{d}$ is nilpotence, viz., $\boldsymbol{d} \circ \boldsymbol{d} = 0$. The proxies of this property in classical vector calculus are the familiar relations

$$\nabla \times (\nabla f) = \mathbf{0}\,, \tag{2.35}$$

$$\nabla \cdot (\nabla \times \mathbf{v}) = 0\,, \tag{2.36}$$

for scalar- and vector-valued functions f and $\mathbf{v}$. Introducing the vector-valued Sobolev spaces

$$H(\text{div}) := \left\{ \mathbf{v} \in \left(L^2\right)^d \text{ s.t. } \|\nabla \cdot \mathbf{v}\|_{L^2} < \infty \right\}\,, \tag{2.37}$$

$$H(\text{curl}) := \left\{ \mathbf{v} \in \left(L^2\right)^d \text{ s.t. } \|\nabla \times \mathbf{v}\|_{L^2} < \infty \right\} \tag{2.38}$$

(in whose company one might also consider referring to H^1 as "H(grad)"), we can depict the relations (2.35)–(2.36) as the **de Rham complex**

$$H^1 \xrightarrow{\nabla} H(\text{curl}) \xrightarrow{\nabla\times} H(\text{div}) \xrightarrow{\nabla\cdot} L^2\,, \tag{2.39}$$

where the image of each overset operator is contained in the kernel of the subsequent one (cf. (2.35)–(2.36)). Sequences of spaces and operators with this property are called **complexes**. If the image of each operator is equal to the kernel of its successor, the complex is said to be **exact**, which is true of the de Rham complex on domains without holes, where every irrotational vector field is the gradient of some scalar potential, and every solenoidal field is the curl of some vector potential.

It is possible to construct finite element (or also isogeometric) subspaces that are related within a corresponding **subcomplex**. The classical finite element spaces used for the discrete H^1, H(curl), H(div), and L^2 spaces polynomially-complete up to degree k are CG_k, **Nédélec** (N_k), **Raviart–Thomas** (RT_k), and DG_k spaces, respectively:[4]

$$\text{CG}_k \xrightarrow{\nabla} \text{N}_{k-1} \xrightarrow{\nabla\times} \text{RT}_{k-1} \xrightarrow{\nabla\cdot} \text{DG}_{k-1}\,. \tag{2.40}$$

Appropriate use of such spaces can lead to numerical methods that *exactly* satisfy the conservation laws of the underlying physics (in a pointwise sense), and is a particularly-popular approach to solving Maxwell's equations with finite elements. In Nédélec spaces, the tangential components of discrete fields are continuous across element boundaries, while, in Raviart–Thomas spaces, the normal components are. The shape functions for discrete de Rham complexes must be pushed forward via **Piola transforms** that are vector calculus proxies of tensor pushforwards applied to the differential forms of exterior calculus.[5] Isogeometric generalizations of these finite element spaces allow for discrete subspaces of $H^1 \cap H$(div/curl), which may be useful for developing conforming variational methods for some problems. Later in the class, we shall revisit this topic in more detail, with concrete examples from continuum mechanics.

[4]References are not consistent with regard to the numbering of RT and N spaces, with some instead designating the RT/N_k spaces as those complete to degree $k-1$.

[5]The divergence-conserving Piola transform of vector fields representing 2-forms is widely known as "the Piola transform" in continuum mechanics and, historically, predates exterior calculus. Chapter 4 covers that formula in detail.

2.5 Finite difference and volume methods

As mentioned in Section 2.1, the variational FEM framework—if understood in an abstract way—is sufficiently-versatile to generate other popular numerical methods, such as the finite differences and volumes, as special cases. The philosophy followed here is that the most principled approach to coupled problems is to "translate" the methods used for all subproblems into the common language of variational forms, to the extent possible. This section considers some examples.

2.5.1 Finite differences from continuous Galerkin

We outline a simple case of 1D finite elements on a uniform grid which reproduces classical finite difference methods. We shall skip over routine evaluations of polynomial integrals, which are left for the reader to complete in Exercise 3.

The **finite difference method (FDM)** basically consists of plugging the PDE solution into the limit definition of a derivative, but not actually taking the limit:

$$\partial_x u(x^n) = \lim_{h\to 0} \frac{u(x^n) - u(x^n - h)}{h} \quad \approx \quad \frac{u^n - u^{n-1}}{h} \, , \tag{2.41}$$

where u^n and u^{n-1} are approximations of $u(x^n)$ and $u(x^n - h)$. FD formulas can be obtained for higher derivatives as well, e.g.,

$$\partial_x^2 u \approx \frac{\partial_x u(x^{n+1}) - \partial_x u(x^n)}{h} \approx \frac{\left(\frac{u^{n+1}-u^n}{h}\right) - \left(\frac{u^n-u^{n-1}}{h}\right)}{h} = \frac{u^{n+1} - 2u^n + u^{n-1}}{h^2} \, . \tag{2.42}$$

However, we can also derive the FDM from the FEM. For example, consider 1D steady heat conduction:

$$-\partial_x^2 u = f \, , \tag{2.43}$$

on the interval 0 to 1, with $u(0) = u(1) = 0$. Suppose we use CG$_1$ nodal basis functions (i.e., a hat function of unit height centered at each node), defined on elements of size h between nodes located at $\{x_1, \ldots, x_N\}$. Evaluating all the integrals, under the assumption that f is a constant, this is equivalent to the FDM

$$-\left(\frac{u^{n+1} - 2u^n + u^{n-1}}{h^2}\right) = f(x^n) \, . \tag{2.44}$$

In the case where f is not constant, one would need to assume that integrals involving it are approximated with a reduced quadrature rule, in which f is evaluated at nodes, with appropriate weights.

As an additional example relevant to fluid dynamics, consider a different equation, advection–diffusion, with another term:

$$-\kappa\partial_x^2 u + a\partial_x u = f \, , \tag{2.45}$$

where $\kappa > 0$ and $a > 0$ are constant coefficients and we use the same boundary conditions as before. If we again discretize this equation using Galerkin's method with a CG$_1$ space, we get the **centered difference** formula

$$a\partial_x u \approx a\left(\frac{u^{n+1} - u^{n-1}}{2h}\right) \tag{2.46}$$

for the additional term. Suppose we instead used a one-sided difference like (2.41) instead of what we got from Galerkin's method, i.e., we use the FDM

$$-\kappa\left(\frac{u^{n+1}-2u^n+u^{n-1}}{h^2}\right)+a\left(\frac{u^n-u^{n-1}}{h}\right)=f(x^n)\ . \tag{2.47}$$

This is referred to as an **upwind difference**. With the same caveats regarding f from before, this is equivalent to applying Galerkin's method to a **modified equation**,

$$-\kappa\left(1+\frac{ah}{2\kappa}\right)\partial_x^2 u+a\partial_x u=f\ . \tag{2.48}$$

in which, essentially, a mesh-dependent **artificial diffusion** has been added. This is an example of a weakly-consistent FEM. From the standpoint of the Lax–Milgram theorem, the artificial diffusion obviously increases the coercivity of the method in H^1, which, if we weren't introducing a consistency error, would be beneficial in the error estimation. We will see later, in Chapter 6, how a similar effect on coercivity can be obtained *in a strongly-consistent (and high-order accurate) manner*, using a Petrov–Galerkin method with the interpretation of coupling the finite element solution to a formal representation of unresolved fine-scale solution features.

Remark 27. With additional bookkeeping and more tedious integrals, the discussion from this section extends to higher-dimensional finite elements/differences on tensor-product grids.

2.5.2 Finite volumes from discontinuous Galerkin

Let us consider here the prototypical problem of pure advection, on which the finite volume (FV) method is frequently illustrated: Given source term $f:(0,1)\to\mathbb{R}$ and boundary data $g\in\mathbb{R}$, find $u:[0,1)\to\mathbb{R}$ such that

$$\partial_x u(x)=f(x)\quad\forall x\in(0,1)\ , \tag{2.49}$$

$$u(0)=g\ . \tag{2.50}$$

This is a simple conservation law, in which the flux is u. The finite volume method is to break $(0,1)$ into cells $\{\Omega_i\}$, which, in 1D, are the intervals $\Omega_i=(x_{i-1/2},x_{i+1/2})$, and pose the integral form of the conservation law on each of these cells, using the divergence theorem (or, in 1D, the fundamental theorem of calculus):

$$u_{i+1/2}-u_{i-1/2}=\int_{x_{i-1/2}}^{x_{i+1/2}}f(x)\,dx\ , \tag{2.51}$$

where $u_{i+1/2}$ and $u_{i-1/2}$ are reconstructed **numerical fluxes** at the faces of cell Ω_i.

However, these are the *same* algebraic equations one obtains from the following variational procedure. Consider multiplying (2.49) by a test function, v, integrating over the domain, and using integration by parts on each cell (which we now call an element):

$$\int_0^1(\partial_x u)v\,dx=\int_0^1 fv\,dx \tag{2.52}$$

$$\Rightarrow\sum_{i=1}^{N_{\text{el}}}\left(-\int_{\Omega_i}u\partial_x v\,dx+uv\Big|_{x_{i-1/2}}^{x_{i+1/2}}\right)=\int_0^1 fv\,dx\ . \tag{2.53}$$

Posing this weak problem over a DG_0 space, i.e., a single constant shape function on each element, we must reconstruct the flux u at $x_{i\pm 1/2}$, since it is not well-defined at these points of discontinuity. As in the FV case, we introduce numerical fluxes $u_{i\pm 1/2}$, and, eliminating terms that are zero for this simple choice of FE space, arrive exactly at the algebraic system (2.51).

However, viewing the problem within the variational framework gives us criteria for designing stable procedures to reconstruct the numerical fluxes $u_{i\pm 1/2}$. For example, the choice of upwind fluxes is equivalent to the upwind FD formula (2.47), which, again leveraging the connection to a variational principle, we see adds coercivity, in the equivalent CG discretization of (2.48), improving stability of the discrete problem. Further, the DG viewpoint suggests a natural extension of FV to higher-order accurate methods which satisfy cell-wise local conservation, by considering DG formulations with higher-order shape functions on each cell/element.

Remark 28. As with the discussion of FD methods, this simple example of equivalence between DG and FV methods can be extended in a conceptually-similar manner to more complicated conservation laws in multiple space dimensions.

2.6 Further reading

As noted in Section 2.1, this chapter provides only a high-level summary of the FEM. Some more detailed references include the following.

- The textbook by Hughes [4] covers the implementation of finite element methods for solid and structural problems in detail, along with aspects of the theory, and is widely praised by the engineering community for its clarity.
- The text by Johnson [13] is somewhat more concise, prioritizing mathematical topics and providing more discussion on FEA of hyperbolic PDEs. Johnson was influential in the early development of FEniCS, and the notation of his text is mirrored by much of the software's documentation.
- A concise undergraduate-level introduction to FEA theory is given by Becker, Carey, and Oden [14].
- A classic reference on the mathematical theory of FEA is that of Strang and Fix [15].
- A recent book by Gockenbach [16] includes modern MATLAB-based code examples and thorough discussion of linear solvers for the systems of equations generated by FE assembly.
- Demkowicz [17] provides thorough, clear, and rigorous notes on the inf-sup condition and its implications for FEA.
- There are of course also textbooks on more specific topics, such as discontinuous Galerkin [18], isogeometric analysis [19], de Rham complex elements [20], and FE-based computational techniques for the model coupled problem of Part II, fluid–structure interaction [1]. These books should be viewed as more advanced references, requiring substantial prior background in the fundamentals of FEA to benefit from.

However, none of these references are required to follow the remaining chapters.

2.7 Exercises

1. Assuming the bilinear form a satisfies the coercivity condition of the Lax–Milgram theorem, derive a sharper version of (2.23) without the "+1" in the constant. (Defining a new constant $C_{\text{new}} := C_B/C_C + 1$ to absorb the "+1" is *not* considered a valid solution.)

   Hint: Start a chain of inequalities off with coercivity, $||e||^2 \leq \frac{1}{C_C} a(e,e) = \cdots$, and try to reach a constant multiple of $||\eta||\,||e||$.

2. (Adapted from [14, Exercise 1.8.2b].) Suppose that we break up the interval $\Omega = (0,1)$ into 1D elements of length h, and define a continuous Galerkin space V^h of polynomial order one. For $u \in C^2(\Omega)$, can we choose $w^h \in V^h$ such that $||u - w^h||_{H^1(\Omega)} \to 0$ as $h \to 0$? The key to answering this question is Taylor's theorem: For some function f, we have

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + R_k(x)\,, \tag{2.54}$$

   where the *remainder*, $R_k(x)$, can be formulated

$$R_k(x) = \frac{f^{(k+1)}(\xi)}{(k+1)!}(x-a)^{k+1}\,, \tag{2.55}$$

   in which ξ is some point between x and a.

   (a) Let $w^h \in V^h$ exactly interpolate u at nodes. How big can the difference $\eta(x) = u(x) - w^h(x)$ get, at some point x between x_A and x_B? Use Taylor's theorem with $f = \eta$, $a = x_A$, and $k = 1$ to get a second-order approximation of $\eta(x)$ plus a remainder, and use the definition of η to write this approximation in terms of u and w^h.

   (b) Using the fact that w^h is a straight line and is equal to u at the endpoints of the element, re-write $\left(w^h\right)'$ in the Taylor series for $\eta(x)$ in terms of $u(x_A)$ and $u(x_B)$.

   (c) Show that $|\eta(x)| \leq Ch^2$.

   Hint: Re-write $u(x_B)$ by using Taylor's theorem, with $f = u$, $a = x_A$, and $k = 1$. Recall that $(x_B - x_A) = h$.

   (d) Apply a similar argument to show that $|\eta'(x)| \leq Ch$. (We follow the convention here that C can represent different constants in different places.)

   Hint: Apply Taylor's theorem with $f = \eta'$, $a = x_A$, and $k = 1$. Recall that, since w^h is a linear interpolant, $\left(w^h\right)'' = 0$ within elements.

   (e) Use these bounds to show that $||\eta||^2_{H^1(\Omega)} \leq Ch^2$.

3. Consider the example from Section 2.5.1, equating FE and finite difference methods.

   (a) Fill in the missing steps in the derivations. In particular, carry out the integrations needed to show how Galerkin's method leads to (2.44) and (2.46), and how Galerkin's method applied to the modified equation (2.48) leads to (2.47).

(b) In the discretization of (2.45), why it would be a bad idea to use a *downwind* finite difference of the form

$$\partial_x u(x^n) \approx \frac{u^{n+1} - u^n}{h} \tag{2.56}$$

in the advection term? Consider, in particular, the case where $ah/\kappa \gg 1$.

4. Consider the unsteady heat equation,

$$\partial_t u(\mathbf{x}, t) - \Delta u(\mathbf{x}, t) = f(\mathbf{x}, t) , \tag{2.57}$$

on a domain Ω, with boundary condition $u(\mathbf{x}, t) = 0$ on $\partial\Omega$ and initial temperature distribution $u(\mathbf{x}, 0) = u_0(\mathbf{x})$.

(a) Derive a "semi-discrete"[6] finite element method (in terms of some generic set of basis functions $\{\phi_1(\mathbf{x}), \dots, \phi_N(\mathbf{x})\}$) of the form

$$\mathbf{M}\left(\frac{d}{dt}\mathbf{U}\right) + \mathbf{KU} = \mathbf{F} , \tag{2.58}$$

where $\mathbf{U}$ is a vector of N unknown coefficients, each of which depend on time, $\mathbf{M}$ and $\mathbf{K}$ are $N \times N$ matrices, and $\mathbf{F}$ is a vector with N entries that does not depend on $\mathbf{U}$.

(b) The problem (2.58) is a system of N ordinary differential equations (ODEs) for the N unknown functions $\{U_1(t), \dots U_N(t)\}$. For ODE systems of the form

$$\frac{d}{dt}\mathbf{X}(t) = \mathbf{f}(\mathbf{X}(t)) , \tag{2.59}$$

where $\mathbf{X}(t)$ is an unknown (vector-valued) function of t and $\mathbf{f}$ is a known function, one possible approximate solution method is the **backward Euler** finite difference scheme,

$$\left(\frac{\mathbf{X}^{n+1} - \mathbf{X}^n}{\Delta t}\right) = \mathbf{f}\left(\mathbf{X}^{n+1}\right) , \tag{2.60}$$

where we assume that $\mathbf{X}^n$ is known and we are looking for the result at the next time step, $\mathbf{X}^{n+1}$. Write a formula analogous to (2.60) for the ODE system (2.58).[7]

(c) What matrix needs to be inverted to get $\mathbf{U}^{n+1}$, given $\mathbf{U}^n$?

5. In a programming language of your choice, *without* using FEniCS, write a 1D CG$_1$ FE code for the problem: Find $u : [0, 1] \to \mathbb{R}$ such that

$$-u''(x) = 1 \quad \forall x \in (0, 1) , \tag{2.61}$$

$$u(0) = u(1) = 0 , \tag{2.62}$$

and verify that it converges at the expected rate in $H^1_0((0, 1))$ as the mesh is refined. *Hint:* One may find it expedient to use the equivalent FDM (cf. Section 2.5.1).

[6]The term "semi-discrete" refers to the fact that we are discretizing in space, but not in time.

[7]A natural question to ask, given the discussion in Section 2.5.1 is: Does this FD in time correspond to an FE time discretization, leading to a space–time FEM? The answer is "yes", but we shall not cover space–time FEA here. Reference [1] contains detailed discussion on a DG-in-time class of space–time methods.

6. In a correct solution to Exercise 5, the discrete solution u^h will actually interpolate the exact solution u at nodes. Prove this property.

   Hint: Construct a specific test function at each node, which increases linearly from zero at the boundaries to one at the node.

   Comment: This is *not* a general property of Galerkin methods; it is peculiar to H^1 seminorm projection.

Chapter 3

Implementation using FEniCS

3.1 Introduction

Computational examples in this class will use open-source software from the FEniCS Project [21, 22] to rapidly prototype solvers for a variety of PDE systems. FEniCS is a system largely conceived within the applied math community, and its usage relies heavily on the abstract functional analytic view of FEA that we have been discussing so far. However, in recent years, FEniCS has seen increasing use by scientific and engineering practitioners interested in complex multiphysics problems that are difficult (at best) to solve with existing analysis software products, most of which are tailored to specific physical domains, and have only limited or awkward mechanisms for solving new user-defined problems.

The project's goal of being suitable for all PDE systems is extremely ambitious, and the current implementation of course falls short in some respects. For instance, while anything is technically possible through advanced use of FEniCS's flexible application programming interface (API), the software's canonical workflow presents no easy way to implement various common features of solid mechanics analyses, such as plasticity with non-radial return mappings, constitutive models involving spectral decomposition of tensors, static condensation tricks, contact mechanics, and many other things that are often simplified-out from PDE models studied by the applied math community. FEniCS nonetheless represents the state-of-the-art in unifying numerical PDE technologies across problem domains; that is why it has been selected for use in the present exposition, which is intended to be useful for researchers across a wide range of disciplines.

A valid criticism of using FEniCS for FEA education is that its standard workflow hides the data structures and algorithms used in FE computations, and is therefore not useful for teaching those aspects of FEA. As we have discussed earlier, though, the emphasis of the current presentation is on formulating complex multiphysics problems so that they can be readily solved using such machinery. Low-level details of a typical FEA implementation can be found in many references, e.g., the textbook [4].

Due to the popularity of FEniCS, a great many high-quality resources already exist for learning it, so the present chapter will not attempt to reinvent the wheel and provide yet another self-contained tutorial. Instead, we provide here some high-level information, links to useful resources, and miscellaneous technical tips that may be difficult to find via Google search. Given the relevant mathematical background, the most effective way to learn FEniCS is to just start using it and look up information and examples as needed; the exercises in Section 3.6 provide some opportunities to practice.

3.2 What is FEniCS?

Development of FEniCS began in 2003, as a collaboration between Chalmers University, in Sweden, and the University of Chicago, in the US. It now involves developers from all over the world, and the project's governance maintains institutional neutrality; the Steering Council of core developers is not affiliated with a single laboratory or country. The name "FEniCS" does not have a single accepted etymology, although it's generally thought that the "FE" stands for "finite elements" and the "CS" stands for "computer science", while the "ni" provides a pronounceable name and mascot.[1]

The FEniCS Project encompasses several open-source software elements mainly designed to be used together, for the solution of PDE systems by the finite element method. The components that most end-users interact with directly are:

- **Unified Form Language (UFL) [23]:** A collection of classes and functions in the Python programming language which provide a domain-specific language for stating variational problems. UFL's structure corresponds closely to the functional analysis concepts and abstract view of FEA introduced in Chapters 1 and 2. UFL includes various algorithms for manipulating variational forms symbolically, including automatic computation of Gateaux derivatives. UFL is also used by the Firedrake Project [24], which maintains Python API compatibility with FEniCS.

- **Dynamic Object-oriented Library for FINite element computation (DOLFIN) [2]:** A C++ finite element library, which interfaces with a variety of external libraries for parallel linear algebra, and other more generic functionality. DOLFIN could, in principle, be used without the rest of the FEniCS toolchain, in a manner similar to other FE libraries like deal.ii, although this is rarely done. Much of DOLFIN's C++ API is available through Python bindings, allowing DOLFIN functionality to be intermixed with UFL problem definitions in Python scripts, which is the most common method of using FEniCS.

- **mshr:** A basic system for generating meshes within Python scripts based on constructive solid geometry operations. While mshr is convenient for quick demonstrations, control over meshing is limited, and it is recommended to instead use an external mesh generator and preprocessor for more complicated problems; see Section 3.3.6 for additional notes and resources.

In a typical workflow, the remaining core components of the project are largely invoked behind the scenes as Python scripts execute:

- **FEniCS Form Compiler (FFC) [25]:** Much like a C++ compiler converts descriptions of algorithms in C++ into low-level machine code for a computer, FFC compiles high-level UFL descriptions of variational forms into efficient C++ routines that plug into DOLFIN.

[1]The University of Chicago's mascot is incidentally "Phil the Phoenix".

- **FInite element Automatic Tabulator (FIAT) [26]:** This component tabulates shape functions for a wide variety of finite element types, up to arbitrarily-high polynomial degrees.

- **dijitso:** A Python library for just-in-time (JIT) compilation of C++ code, to permit the use of new C++ routines generated at run-time, such as the output of FFC or custom user-provided code snippets.

As explained by the current development roadmap [27], a major redesign of the DOLFIN and FFC components is currently underway in the experimental DOLFINx and FFCx repositories, which are the focus of most ongoing development. The updated toolchain is collectively called “FEniCSx”. FEniCSx is recommended for most new FEniCS-based research projects. However, distribution and corrective maintenance of the current stable version, referred to as “Legacy FEniCS” (or sometimes “Old FEniCS”), is planned to continue for the foreseeable future.

3.3 Useful information

As noted in Section 3.1, it would be redundant to prepare yet another self-contained tutorial on FEniCS, as many are freely available online. However, this section compiles into one place various helpful notes and references to sources for additional practical information relating to FEniCS usage.

3.3.1 Python

The canonical workflow for using FEniCS involves writing scripts in Python that define variational problems in UFL, then invoking Python bindings of DOLFIN functions to assemble and solve systems of equations. Thus, it is essential to understand basic Python syntax. Fortunately, Python is one of the easiest programming languages to learn (or at least reach the proficiency level needed to write useful programs). Readers familiar with MATLAB (and the general feature set to expect of a modern programming language) can likely pick up the necessary elements of Python by simply diving directly into FEniCS examples and Google-searching specific points as needed (e.g., “how to write text to file in python”). As such, dedicated study of general-purpose Python tutorials is not recommended for the purposes of quick-starting FEniCS usage. However, a few potentially-confusing points for engineering students with only MATLAB programming experience are:

- **Zero-based indexing:** Like C/C++, Java, and most other popular general-purpose programming languages, array-like data types are indexed starting from zero, with square brackets, not starting from one with parentheses, as in some scientifically-focused languages like MATLAB and Fortran.

- **Whitespace significance:** The end of `if`-blocks, function bodies, etc., is signaled by a decrease in indentation level rather than an `end` (as in Matlab) or a closing curly brace (as in C/C++ or Java). Likewise the beginning of such blocks is signaled by an increase in indentation level. Thus, the indentation habits that are *recommended* with most languages are *required* in Python.

- **Operator overloading:** This functionality actually does exist in MATLAB, but most engineering users are unaware it. However, operator overloading is used extensivly by FEniCS (particularly UFL). Basically, it is the mapping of operators (like `+`, `-`, `*`, etc.) to custom methods of user-defined data types (`classes`) which are called when those data types are passed as operands. This is what UFL uses to enable near-mathematical notation like `form = integrand*dx`, where `form` is an instance of a class representing a variational form, `integrand` refers to a syntax tree for some mathematical expression, and `dx` is an integration measure for some domain. In this case, the action of the * operator has been defined for non-numerical objects from UFL.

- **Incompatible versions:** There are two widely-used versions of Python, namely Python 2 and Python 3. The transition from 2 to 3 broke backward compatibility, and, thus, to support legacy scripts, the default Python interpreter on most systems (i.e., what runs when entering "`python`" in a terminal) is still Python 2. However, most ongoing Python software development has transitioned to Python 3, including the FEniCS Project, which no longer supports Python 2. The Python 3 interpreter is usually invoked via `python3` in command line environments. **FEniCS scripts must be run using Python 3**.

- **Proliferation of development environments:** Most MATLAB users work within the integrated development environment (IDE) that ships with the software. By contrast, Python developers use a wide variety of workflows. There are many popular IDEs, with feature sets that may suit individual users, but one can also work productively using a general-purpose text editor (e.g., Notepad++, gedit, Vim, Emacs, etc.) to write code and a system command prompt to execute it.

Python ranks, by most measures, among the most popular programming languages in the world, and many online resources for setting it up on different types of computer systems are readily available.

3.3.2 Installation

There are a variety of ways to install and use FEniCS:

- The most convenient platform on which to get FEniCS running is **Ubuntu Linux**, for which personal package archives (PPAs) are maintained. Reliable installation instructions are readily available on the FEniCS homepage [28]. The same instructions can be followed within Windows Subsystem for Linux on **Windows 10**.

- **Docker** is a containerization software that allows self-contained software environments to be distributed and run on **any system** for which Docker is supported. Docker containers for different versions of FEniCS are maintained, and this is the recommended method of using FEniCS on personal computers not running Ubuntu Linux. Containerization is also useful for working with different versions of FEniCS simultaneously on one system. Unfortunately, there is a **nontrivial learning curve** associated with Docker usage and some troubleshooting is almost inevitable. Using Docker is nonetheless much easier than attempting to build FEniCS from source, due to FEniCS's complex design and host of dependencies.

- **Singularity** is a containerization software tailored to high-performance computing clusters. This is the recommended method of **using FEniCS on clusters**. A Singularity image can be created from any Docker container. As with Docker, some system-specific troubleshooting is almost always necessary, yet still preferable to building from source.

3.3.3 Finding current documentation

An important point to keep in mind when searching for FEniCS documentation and examples is that the API has changed over time, and **many of the top search results from Google will be based on (sometimes much-)older versions** of the software. Links to the most current API documentation and demos can be found on the FEniCS homepage [29]. It may also be helpful to include the current year (in quotes) in Google search queries related to specific functionality.

3.3.4 Some tutorials

The recommended starting point for learning about FEniCS usage is the collection of documented demos distributed with DOLFIN. Some additional high-quality third-party tutorials include:

- Documented examples focused on solid and structural mechanics, by Bleyer [30].

- A book with examples from many different physical systems, by Abali [31].

Some further examples related to fluids, solids, and fluid–structure interaction will be provided later, in Part II.

3.3.5 Online community

A major advantage of FEniCS is its large and active user community. The main online forum for discussion of FEniCS *usage* is the **Discourse group** [32]. Discussion on *development* (of FEniCS itself, not applications using it) mainly occurs on the **Slack channel** [33]. Bugs in FEniCS's components are reported through their respective **issue trackers on Bitbucket**. Some points to keep in mind when posting to the Discourse group are:

- Discourse uses the Markdown language to format text in posts, which is especially useful for ensuring that code is readable. Information on Markdown syntax is readily available online. The main points to be aware of are that LaTeX math syntax should be enclosed in dollar signs, and code snippets (or other verbatim text) should be enclosed in backticks.

- When possible, try to include a **minimal working example (MWE)** of code that other users can run to reproduce the problem encountered. Simplifying application code into an MWE is itself a good debugging exercise, and may often reveal the problem without a need for outside help. Preparing an MWE is also part of best practices for reporting bugs.

In the context of this class, students should feel free to ask for general help with FEniCS from the user community, but **should not** directly pose homework exercises as questions to online forums.

3.3.6 Generating and importing meshes

For simplicity and ease of distribution, examples in these notes will primarily use FEniCS's basic built-in mesh generation capabilities (i.e., `UnitSquareMesh`, `UnitCubeMesh`, etc., and mshr). However, DOLFIN has fairly robust support for importing (primarily simplicial) meshes generated by third-party software. Some useful references are itemized below:

- Perhaps the most widely used external mesh generator for FEniCS is the open-source software **Gmsh** (usually pronounced "gee mesh") [34]. Note that Gmsh scripting can be done directly in Python programs via **pygmsh** [35].
- The open-source Python library **meshio** [36] can be used to convert between many different mesh formats, including those that can be read by DOLFIN. (The older FEniCS utility program called dolfin-convert has now been largely deprecated in favor of meshio.)
- A FEniCS workflow using the open-source graphical preprocessor **SALOME** is described in [31, Appendix A.3].
- Some initial support for spline-based isogeometric models is available through the library **tIGAr** [37], although there are currently only limited software tools available for creating analysis suitable geometries.

3.3.7 Visualizing multi-field results in ParaView

While interactive visualization of results via Matplotlib can be convenient for small example programs, visualization for more complicated FEniCS applications is usually performed with the open-source program **ParaView** [38]. Some points to consider when using ParaView to visualize FEniCS output are:

- The automatically-generated names of `Function`s in FEniCS are not necessarily consistent between runs of the same program. However, the `Function` names need to be consistent to re-use ParaView state files, which save (sometimes tedious) data manipulations. For programs whose output requires nontrivial ParaView states, it is most convenient to manually rename `Function`s, like

```
u = Function(V)
# ...
u.rename("u","u")
File("u.pvd") << u
```

- In coupled problems, one often has multiple different fields defined in different `FunctionSpace`s on the same mesh. However, if these fields are written to separate ParaView files, they will be on identical but logically-separate meshes during visualization. The somewhat-tedious procedure to get all data on one mesh is as follows:
  - Load the files corresponding to different fields and click on the "Apply" button.
  - Select them all in the Pipeline Browser.

  - Create an “Append Attributes” filter merging them (which can be found via `Filters > Search ...`) and click the “Apply” button.

  Make sure to **save the ParaView state file** (`.pvsm`) after these steps, to avoid repeating them.

- To deform a mesh by some vector-valued displacement field, use a “Warp by Vector” filter. If the vector field is one among several fields merged via “Appended Attributes”, then the other fields can be plotted as contours, isosurfaces, etc. on the deformed mesh.

- To plot use some function of fields written to ParaView files in the visualization process, use a “Calculator” filter. Alternatively, one might consider defining the corresponding expression in UFL, but it would need to be projected to a finite element space to be writable in ParaView format. Note that the built-in `project` function in FEniCS is an L^2 projection, which involves a linear solve, and can be computationally costly. (It also produces bad (oscillatory) results when projecting a discontinuous function to a continuous function space, which surprises some users.) It is *not* the same thing as interpolation. Interpolation is available via the `interpolate` function, but can only be used with a more restricted set of argument types.

3.4 A peek under the hood

The highly-abstract interface to FEniCS makes many engineering users uncomfortable, as it may not be clear exactly what computations are taking place. In this section, we dissect a minimal example of solving a variational problem, to partly illustrate the inner workings of the system. Perhaps the simplest possible problem is $L^2(\Omega)$ projection of some function $f \in L^2(\Omega)$ onto a finite element subspace of $L^2(\Omega)$. For further simplicity, consider the 1D domain $\Omega = (0, 1)$ and let $f \equiv 1$. The problem becomes: Find $u^h \in \mathrm{CG}_1^h$ such that $\forall v^h \in \mathrm{CG}_1^h$,

$$\int_\Omega u^h v^h \, dx = \int_\Omega v^h \, dx \, . \tag{3.1}$$

Using the FEniCS Python API, this can be solved with roughly as many characters as were needed to completely state the problem in LaTeXsource code:

```
from dolfin import *
mesh = UnitIntervalMesh(10)
V = FunctionSpace(mesh,”CG”,1)
u = TrialFunction(V)
v = TestFunction(V)
u_h = Function(V)
solve(u*v*dx==v*dx,u_h)
```

When running, one sees terminal output like

```
Calling FFC just-in-time (JIT) compiler, this may take some time.
Calling FFC just-in-time (JIT) compiler, this may take some time.
Solving linear variational problem.
```

The JIT compilation refers to FFC generating C++ code, which is then used by DOLFIN to assemble and solve the linear system for the unknown degrees of freedom (as announced by the last line of output). If the program is run again, the JIT compilation messages will be absent. This is because the C++ code generated for this problem is cached in the filesystem, to avoid redundant compilation. We can also manually inspect this cache, though, which may be instructive. By default, code and compiled shared-object files are cached in

```
~/.cache/dijitso/
```

Looking under the `src` subdirectory, one sees compressed C++ source files with names like

```
ffc_form_<long hexadecimal string>.cpp.gz
```

The hex string in each file name is a hash of the information that uniquely defines a given form. Extracting and opening the files corresponding to the L^2 projection example above and isolating the interesting part of the code in the forms' `tabulate_tensor` methods, we see the computation of the CG_1 mass matrix

```
alignas(32) static const double FE0_C0_D1_Q2[1][1][2]
  = { { { -1.0, 1.0 } } };
alignas(32) static const double PI0[1][2][2] =
    { { { 0.3333333333333334, 0.1666666666666667 },
        { 0.1666666666666667, 0.3333333333333334 } } };
// Unstructured piecewise computations
const double J_m0_c0
  = coordinate_dofs[0] * FE0_C0_D1_Q2[0][0][0]
  + coordinate_dofs[1] * FE0_C0_D1_Q2[0][0][1];
alignas(32) double sp[1];
sp[0] = std::abs(J_m0_c0);
A[0] = sp[0] * PI0[0][0][0];
A[1] = sp[0] * PI0[0][0][1];
A[2] = sp[0] * PI0[0][1][0];
A[3] = sp[0] * PI0[0][1][1];
```

where `sp[0]` is the element length and scales the integrals of products of shape functions on the reference element (tabulated in the array `PI0`) to fill in the four entries of the local mass matrix, assigned to the array `A`. Likewise similar code can be found in a separate form file for the computation of the local right-hand side vector

```
alignas(32) static const double FE0_C0_D1_Q1[1][1][2]
  = { { { -1.0, 1.0 } } };
alignas(32) static const double PI0[1][2]
  = { { 0.5, 0.5 } };
// Unstructured piecewise computations
const double J_m0_c0
  = coordinate_dofs[0] * FE0_C0_D1_Q1[0][0][0]
  + coordinate_dofs[1] * FE0_C0_D1_Q1[0][0][1];
alignas(32) double sp[1];
```

```
    sp[0] = std::abs(J_m0_c0);
    A[0] = sp[0] * PI0[0][0];
    A[1] = sp[0] * PI0[0][1];
```

whose two entries are assigned to the array `A`, and clearly correspond to integrals of each shape function. The shape functions themselves are implemented in the FIAT component of FEniCS, which is written in Python. However, these Python implementations need only be used once, on the reference element, to generate C++ code as shown above, which is what actually executes in the inner loop over all elements in a mesh.

The action of the `solve` function, when invoked as above, is twofold. First, it assembles the matrix and vector for the linear algebraic problem by executing the above C++ code for each element of the mesh and adding the entries of the local `A` matrices and vectors to the appropriate entries of the global ones. (This could alternatively be performed alone using the `assemble` function.) Second, it calls a linear solver—by default a direct solver, but others are available—to obtain the vector of basis function coefficients for the output `Function`, `u_h`.

3.5 How does FEniCS take derivatives?

The `derivative` function in UFL is very convenient for solving complicated nonlinear problems by Newton's method or related algorithms. However, many users are unsure whether these Gateaux derivatives are taken numerically, e.g., by a finite difference approximation of the definition, or by the procedure of automatic differentiation, where the chain rule is applied to low-level mathematical operations.

Derivatives are computed symbolically, using computer algebra functionality in UFL. It is actually possible to get human-readable expressions for these derivatives, as demonstrated in the following code snippet:

```
from dolfin import *
from ufl import replace
mesh = UnitCubeMesh(1,1,1)
V = FunctionSpace(mesh,"CG",1)
u = Function(V)
u.rename("u","u")
a = (sin(u)**2 + 1)*dot(grad(u),grad(u))
du = TestFunction(V)
deriv_a = derivative(a,u,du)
deriv_a = replace(deriv_a,{du:Function(V,name="du")})
print(deriv_a)
```

The output of this script is

```
    (1 + sin(u) ** 2) * (sum_{i_8} ((grad(u))[i_8] * (grad(du))[i_8] +
    (grad(u))[i_8] * (grad(du))[i_8]) ) + 2 * du * cos(u) * sin(u) *
    (sum_{i_8} (grad(u))[i_8] * (grad(u))[i_8] )
```

which one can easily verify by manual calculation, using the definition of the Gateaux derivative. (Note that the symbol `i_8` in the output is an automatically-generated dummy index for summations.)

3.6 Exercises

1. The following code solves a nonlinear PDE, leveraging UFL's `derivative` function to perform linearization for Newton iteration.

```
from dolfin import *
Nel = 10
mesh = UnitIntervalMesh(Nel)
V = FunctionSpace(mesh,"CG",1)
u = Function(V)
v = TestFunction(V)
x = SpatialCoordinate(mesh)
F = ((u+1)**2)*inner(grad(u),grad(v))*dx + (u*u + u)*v*dx\
    - sin(pi*x[0])*v*dx

# Replace with a bilinear form:
Delta_u = TrialFunction(V)
J = derivative(F,u,Delta_u)

# Replace with a for-loop implementing Newton's method:
solve(F==0,u,J=J)
```

   For this exercise,

   (a) Write the strong form of the PDE problem solved by this code in traditional mathematical notation, including boundary conditions.

   (b) Modify the code to manually implement the Newton iteration as a Python `for` loop, only using `solve` for the linear problem at each Newton step.

   (c) Modify the code to define `J` directly in UFL by deriving the expression for it manually, using the definition of a Gateaux derivative.

2. The **method of manufactured solutions** is a way to systematically create test cases for numerical methods. The idea is to decide what the exact solution should be first, plug it into the strong form of the PDE to derive the corresponding source term, then execute the numerical method with that source term and check the error. With FEniCS's UFL, the often-messy derivation of the source term can be automated by defining the exact solution in terms of

```
x = SpatialCoordinate(mesh) # This is a vector, even in 1D.
```

   and then defining the source term by applying UFL differential operators (e.g., `div`, `grad`, etc.) to it. Apply this workflow to manufacture the solution $u(x) = \sin(\pi x)$ to the problem

$$-\left(\left(u(x)^2 + 1\right) u'(x)\right)' = f(x) \quad \forall x \in (0,1) \tag{3.2}$$

$$u(0) = u(1) = 0 \tag{3.3}$$

and use the log–log plotting technique from Remark 26 to verify that CG_1 approximations converge at optimal rate in $H^1_0((0,1))$. *Hint 1:* To compute the $H^1_0((0,1))$ norm, note that you can integrate expressions with the `assemble` function like

```
    integral = assemble(integrand*dx)
```

where `integral` will be a `float` if `integrand` does not contain any `TestFunctions` or `TrialFunctions`. *Hint 2:* An example of manufacturing a solution to a different problem is given in the code snippet of Exercise 3.

3. With FEniCS, time stepping for unsteady problems is typically implemented manually, using Python control structures. The following code implements unsteady heat conduction using the backward Euler scheme for time integration:

```
from dolfin import *
N = 16
T = 1.0
Dt = Constant(T/N)
mesh = UnitIntervalMesh(N)
V = FunctionSpace(mesh,"CG",1)
u = TrialFunction(V)
uh = Function(V)
uh_old = Function(V)
udot = (u-uh_old)/Dt
v = TestFunction(V)
t = Constant(0.0)
tv = variable(t)
x = SpatialCoordinate(mesh)[0]
u_exact = sin(tv)*sin(pi*x)
udot_exact = diff(u_exact,tv)
f = udot_exact - div(grad(u_exact))
F = ((udot-f)*v + dot(grad(u),grad(v)))*dx
bc = DirichletBC(V,Constant(0.0),"on_boundary")
for step in range(0,N):
    t.assign(float(t)+float(Dt))
    solve(lhs(F)==rhs(F),uh,bcs=[bc,])
    uh_old.assign(uh)
print(sqrt(assemble(((uh-u_exact)**2)*dx)))
```

For a generic ordinary differential equation system

$$\dot{\mathbf{X}}(t) = \mathbf{f}(\mathbf{X}, t)\,, \tag{3.4}$$

the backward Euler algorithm to compute the $n+1$st unknowns from the nth with step size Δt is

$$\frac{\mathbf{X}^{n+1} - \mathbf{X}^n}{\Delta t} = \mathbf{f}\left(\mathbf{X}^{n+1}, t^{n+1}\right)\,. \tag{3.5}$$

Now,

(a) Write the strong PDE problem solved by this code in traditional mathematical notation, including boundary and initial conditions.

(b) Modify the above code to instead use the implicit midpoint rule,

$$\frac{\mathbf{X}^{n+1} - \mathbf{X}^n}{\Delta t} = \mathbf{f}\left(\frac{1}{2}\left(\mathbf{X}^{n+1} + \mathbf{X}^n\right), \frac{1}{2}\left(t^{n+1} + t^n\right)\right) . \tag{3.6}$$

(c) Verify that, after upgrading to implicit midpoint, the L^2 error in the solution at the end of the time interval converges at rate two with respect to $h = C\Delta t$. *Hint:* If only first-order convergence is obtained, check that `t` is always assigned to the right value when used, both inside the time stepping loop (for `f`) and when checking the error (for `u_exact`).

(d) Check the convergence rate of the H^1 seminorm of error at time T for both time integration schemes, using both CG_1 and CG_2 spaces for `V`. Explain why the results are what they are.

4. The following code snippet (finally!) solves our first coupled problem, in which we use a mixed function space to represent several interacting fields:

```
from dolfin import *
N = 16
mesh = UnitCubeMesh(N,N,N)
d = mesh.geometry().dim()
cell = mesh.ufl_cell()
uE = VectorElement("CG",cell,1)
TE = FiniteElement("CG",cell,1)
V = FunctionSpace(mesh,MixedElement([uE,TE]))
def boundary0(x, on_boundary):
    return on_boundary and near(x[0],0)
def boundary1(x, on_boundary):
    return on_boundary and near(x[1],0)
def boundary2(x, on_boundary):
    return on_boundary and near(x[2],0)
bc1 = DirichletBC(V.sub(0).sub(0), Constant(0), boundary0)
bc2 = DirichletBC(V.sub(0).sub(1), Constant(0), boundary1)
bc3 = DirichletBC(V.sub(0).sub(2), Constant(0), boundary2)
bc4 = DirichletBC(V.sub(1), Constant(0), boundary0)
bc5 = DirichletBC(V.sub(1), Constant(1), boundary1)
u,T = TrialFunctions(V)
v,Q = TestFunctions(V)
K = Constant(1.0)
G = Constant(1.0)
alpha = Constant(1.0)
I = Identity(d)
eps = sym(grad(u)) - alpha*I*T
```

```
sigma = K*tr(eps)*I + 2*G*(eps-tr(eps)*I/3)
f = Constant(d*(0,))
k = Constant(1)
a = inner(sigma,grad(v))*dx + dot(k*grad(T),grad(Q))*dx
L = inner(f,v)*dx
uTh = Function(V)
solve(a==L,uTh,[bc1,bc2,bc3,bc4,bc5])
u_out,T_out = uTh.split()
u_out.rename("u","u")
T_out.rename("T","T")
File("u.pvd") << u_out
File("T.pvd") << T_out
```

For this exercise,

(a) Interpret the code. What problem is it solving? Write the PDE system in strong form, using traditional mathematical notation and including all boundary conditions.

(b) Note that the coupling here is *one-way*; modify the program to compute `u` and `T` with separate solves, and verify that the solutions stay the same.

(c) Follow the instructions from Section 3.3.7 to visualize the results in ParaView, with the field `T` as color on a mesh deformed by the field `u`.

II

Examples from fluid–structure interaction

Engineers think in examples, but mathematicians think in counterexamples.

—Ivo M. Babuška

Chapter 4

Continuum mechanics on moving domains

4.1 Introduction

Before we can discuss numerical methods for analyzing **fluid–structure interaction (FSI)**, we must first clearly define the PDE system with which we will model it. A rigorous treatment of FSI requires careful attention to the kinematics of continuous media. Even most graduate courses in fluid and solid mechanics cover only specialized descriptions of motion that are suitable to many—but not all—cases of practical interest in their respective subdisciplines. However, we must introduce the general case to reconcile the different descriptions favored in fluid and solid mechanics communities. We shall not go into such depth on topics specific to the fluid and structure subproblems. Our discussion of fluids will be limited to incompressible flows with Newtonian viscosity laws, our discussion of solid mechanics will be limited to (hyper)elasticity, and thermodynamics will be ignored altogether. The chapter begins, in Section 4.2, with a distillation of the discussion on kinematics and balance laws developed in [39], whose notation we largely follow. Section 4.3 then reviews several examples of specific balance laws relevant to continuum mechanics.

Remark 29. This chapter makes heavy use of both **index notation** and **direct notation** for tensor calculus. These notations, their counterparts in FEniCS's UFL, and some useful formulas are reviewed in Appendix B.

4.2 Kinematics for general balance laws

From a computational (and sometimes physical) perspective, it is often most convenient to refer a PDE problem posed on a deforming domain back to a static reference domain, where time-independent function spaces for a finite element method may be constructed. Let us define this static domain as $\Omega_y \subset \mathbb{R}^d$, where d is the space dimension. Extruding this in the time direction over the interval $\mathcal{I} = (0, T)$, we get space–time **referential domain** $Q_y = \mathcal{I} \times \Omega_y \subset \mathbb{R}^{d+1}$. We now consider a mapping,

$$\tilde{\phi} : Q_y \to Q_x \subset \mathbb{R}^{d+1} , \tag{4.1}$$

where Q_x is the **spatial domain**, representing a subset of physical space–time on which a PDE system will be posed. In particular, we assume that $\tilde{\phi}$ has the form

$$\tilde{\phi}(\mathbf{y}, r) = \begin{pmatrix} t \\ \mathbf{x} \end{pmatrix} \tag{4.2}$$

where $t = r$,

$$\mathbf{x} = \hat{\phi}(\mathbf{y}, r) , \tag{4.3}$$

and $\hat{\phi} : Q_y \to \mathbb{R}^d$ is the spatial part of the mapping $\tilde{\phi}$. This mapping is illustrated in Figure 4.1.

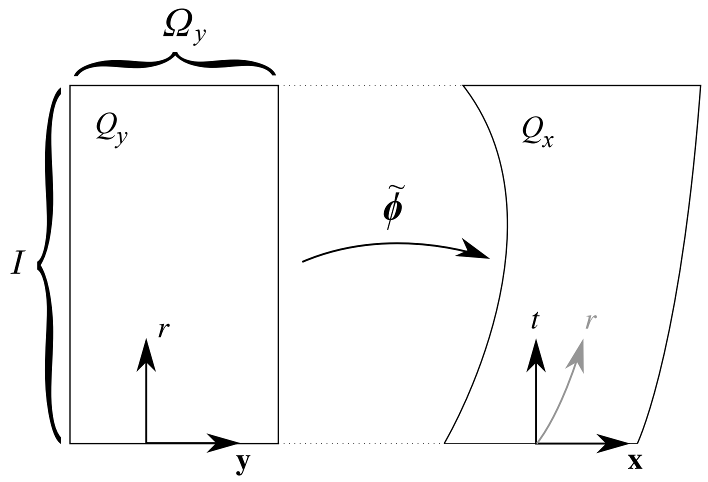

Figure 4.1: Space–time mapping between referential and spatial domains. Consider, in particular, the distinction between the derivatives $\partial/\partial r$ and $\partial/\partial t$ in Q_x.

For clarity, we shall, unless otherwise specified, use the symbols $\mathbf{y}$ and r to refer to a generic point in Q_y and $\mathbf{x}$ and t to refer to a generic point in Q_x. Further, as done above with $\tilde{\phi}$ and $\hat{\phi}$, we will follow the convention of using a tilde, $\tilde{(\cdot)}$, to indicate space–time vectors and tensors defined on the reference domain, and a hat, $\hat{(\cdot)}$, to indicate their spatial components.

Remark 30. Although our assumption on the form of $\tilde{\phi}$ implies that $t = r$, the distinction between referential and spatial time coordinates will become significant when taking partial derivatives with respect to them.

The deformation gradient of $\tilde{\phi}$ is

$$\tilde{\mathbf{F}} := D\tilde{\phi} = \begin{pmatrix} 1 & \mathbf{0}^T \\ \hat{\mathbf{v}} & \hat{\mathbf{F}} \end{pmatrix} , \tag{4.4}$$

where

$$\hat{\mathbf{F}} = \frac{\partial \hat{\phi}}{\partial \mathbf{y}} \tag{4.5}$$

is the deformation gradient of the spatial part of the mapping and $\hat{\mathbf{v}} = \partial_r \hat{\mathbf{u}}$ is the velocity with which the spatial domain deforms, in terms of its displacement $\hat{\mathbf{u}}(\mathbf{y}, r) = \hat{\phi}(\mathbf{y}, r) - \mathbf{y}$ from the referential domain. It is then easy to verify that

$$\tilde{\mathbf{F}}^{-1} = \begin{pmatrix} 1 & \mathbf{0}^T \\ -\mathbf{F}^{-1}\hat{\mathbf{v}} & \hat{\mathbf{F}}^{-1} \end{pmatrix} \tag{4.6}$$

and $\tilde{J} := \det \tilde{\mathbf{F}} = \det \hat{\mathbf{F}} =: \hat{J}$.

We are now equipped to derive the main result used to refer physical conservation laws posed on Q_x to Q_y. The **space–time Piola transform** of a space–time flux vector field $\tilde{\gamma}_x : Q_x \to \mathbb{R}^{d+1}$ is defined as

$$\tilde{\gamma}_y := \tilde{J}\tilde{\mathbf{F}}^{-1}\tilde{\gamma}_x \tag{4.7}$$

and satisfies the property of preserving conservation structure from Q_x on the referential domain:

$$\int_{Q_x} \tilde{\nabla}_x \cdot \tilde{\gamma}_x \, dQ_x = \int_{Q_y} \tilde{\nabla}_y \cdot \tilde{\gamma}_y \, dQ_y \, , \tag{4.8}$$

where

$$\tilde{\nabla}_x := \begin{pmatrix} \partial_t \\ \nabla_x \end{pmatrix} \quad \text{and} \quad \tilde{\nabla}_y := \begin{pmatrix} \partial_r \\ \nabla_y \end{pmatrix} . \tag{4.9}$$

The relation (4.8) follows from the **Piola identity**,

$$\tilde{\nabla}_y \cdot \left(\tilde{J}\tilde{\mathbf{F}}^{-1} \right) = \mathbf{0} \, . \tag{4.10}$$

To derive this, we impose Cartesian space–time coordinate charts $\{x_a\}$ and $\{y_A\}$ on the spatial and referential domains respectively, using lower-case indices in Q_x and upper-case indices in Q_y, and letting indices range from 0 (time) through d (so $t = x_0$ and $r = y_0$). Then the left-hand side of (4.10) becomes

$$\left(\tilde{J}\tilde{F}^{-1}_{Aa} \right)_{,A} = \tilde{J}_{,A}\tilde{F}^{-1}_{Aa} + \tilde{J}\tilde{F}^{-1}_{Aa,A} \, , \tag{4.11}$$

where indices after the comma indicates partial differentiation with respect to coordinates $\{x_a\}$ or $\{y_A\}$, depending on the letter case of the index. The derivative of $\tilde{J}$ in first term on the right-hand side of (4.11) can be expanded using the chain rule:

$$\tilde{J}_{,A} = \frac{\partial \tilde{J}}{\partial \tilde{F}_{bB}} \tilde{F}_{bB,A} = \tilde{J}\tilde{F}^{-T}_{bB}\tilde{\phi}_{b,BA} \, , \tag{4.12}$$

where the second equality invokes the definition of $\tilde{\mathbf{F}}$ and the tensor calculus identity (B.27). For the derivative of $\tilde{\mathbf{F}}^{-1}$ in the second term on the right-hand side of (4.11), we likewise apply the chain rule, another tensor calculus identity (B.26), and the definition of $\tilde{\mathbf{F}}$ to obtain

$$\tilde{F}^{-1}_{Ca,A} = \frac{\partial \tilde{F}^{-1}_{Ca}}{\partial \tilde{F}_{bB}} \tilde{F}_{bB,A} = -\tilde{F}^{-1}_{Cb}\tilde{\phi}_{b,BA}\tilde{F}^{-1}_{Ba} \, , \tag{4.13}$$

which implies

$$\tilde{F}^{-1}_{Aa,A} = -\tilde{F}^{-1}_{Ab}\tilde{\phi}_{b,BA}\tilde{F}^{-1}_{Ba} \tag{4.14}$$

by contraction over indices A and C. Substituting (4.12) and (4.14) back into the right-hand side of (4.11), we recover the Piola identity (4.10):

$$\left(\tilde{J}\tilde{F}^{-1}_{Aa} \right)_{,A} = \tilde{J}\tilde{\phi}_{b,AB} \left(\tilde{F}^{-1}_{Bb}\tilde{F}^{-1}_{Aa} - \tilde{F}^{-1}_{Ab}\tilde{F}^{-1}_{Ba} \right) = 0 \, , \tag{4.15}$$

where the last equality follows from the symmetry of $\tilde{\phi}_{b,AB}$ and anti-symmetry of the parenthesized terms in A and B.

Physical conservation laws are often naturally formulated with respect to a specific preferred referential domain, called the **material description**, in which we make the notational substitutions $\mathbf{y} \to \mathbf{X}$ and $r \to s$, to distinguish it from other arbitrary choices of referential configuration. Further, we omit hats from the spatial components of vector and tensor fields defined on Q_X, to recover standard notations from finite-strain solid mechanics. In continuum mechanics, the material description's referential domain has the interpretation that every point in Ω_X corresponds to a single material point for all time. Taking Ω_x to be the image of Ω_X by the mapping ϕ, then a typical conservation law for a scalar field α would be

$$\frac{d}{dt}\int_{\Omega_x} \alpha\, d\Omega_x = -\int_{\partial\Omega_x} \gamma_x \cdot \mathbf{n}_x\, d\partial\Omega_x + \int_{\Omega_x} \beta\, d\Omega_x\,, \tag{4.16}$$

where γ_x is the spatial flux, $\mathbf{n}_x$ is the outward-facing normal to $\partial\Omega_x$, and β is a source term. Because Ω_x varies in time, though, we must apply the **Reynolds transport theorem** to bring d/dt inside of the integral on the left-hand side of (4.16) as a partial derivative:

$$\begin{aligned}
\frac{d}{dt}\int_{\Omega_x} \alpha\, d\Omega_x &= \int_{\Omega_X} \partial_s(\alpha J)\, d\Omega_X && (4.17)\\
&= \int_{\Omega_X} (\partial_t\alpha + \mathbf{v}\cdot\nabla_x\alpha) J\, d\Omega_X + \int_{\Omega_X} \alpha(\partial_s J)\, d\Omega_X && (4.18)\\
&= \int_{\Omega_x} \partial_t\alpha\, d\Omega_x + \int_{\Omega_x} \mathbf{v}\cdot\nabla_x\alpha\, d\Omega_x + \int_{\Omega_x} \alpha\nabla_x\cdot\mathbf{v}\, d\Omega_x && (4.19)\\
&= \int_{\Omega_x} \partial_t\alpha\, d\Omega_x + \int_{\Omega_x} \nabla_x\cdot(\alpha\mathbf{v})\, d\Omega_x && (4.20)\\
&= \int_{\Omega_x} \partial_t\alpha\, d\Omega_x + \int_{\partial\Omega_x} \alpha\mathbf{v}\cdot\mathbf{n}_x\, d\partial\Omega_x\,, && (4.21)
\end{aligned}$$

where we have used

$$\partial_s J = \frac{\partial J}{\partial F_{iJ}}\frac{\partial F_{iJ}}{\partial s} = JF^{-1}_{Ji}\frac{\partial^2 x_i}{\partial s\partial X_J} = JF^{-1}_{Ji}\frac{\partial v_i}{\partial X_J} = J\frac{\partial v_i}{\partial X_J}\frac{\partial X_J}{\partial x_i} = J\nabla_x\cdot\mathbf{v} \tag{4.22}$$

(invoking (B.27) for the second equality) to obtain (4.19). Returning to (4.16), this results in

$$\int_{\Omega_x} \partial_t\alpha\, d\Omega_x = -\int_{\partial\Omega_x} (\gamma_x + \alpha\mathbf{v})\cdot\mathbf{n}_x\, d\partial\Omega_x + \int_{\Omega_x} \beta\, d\Omega_x\,. \tag{4.23}$$

Applying the divergence theorem, then integrating in time, we get a conservation law in space–time:

$$\int_{Q_x} \partial_t\alpha + \nabla_x\cdot(\alpha\mathbf{v} + \gamma_x) - \beta\, dQ_x = \int_{Q_x} \tilde{\nabla}_x\cdot\tilde{\gamma}_x - \beta\, dQ_x = 0\,, \tag{4.24}$$

where the space–time flux $\tilde{\gamma}_x$ is

$$\tilde{\gamma}_x := \begin{pmatrix} \alpha \\ \alpha\mathbf{v} + \gamma_x \end{pmatrix}. \tag{4.25}$$

Using the identity (4.8), we then have, for some arbitrary referential configuration Q_y, that

$$\int_{Q_y} \tilde{\nabla}_y \cdot \left(\tilde{J}\tilde{\mathbf{F}}^{-1}\tilde{\gamma}_x\right) - \tilde{J}\beta \, dQ_y = 0 \,, \tag{4.26}$$

which, expanding in terms of spatial components (using (4.6)) and localizing in time gives

$$\int_{\Omega_y} \partial_r(\hat{J}\alpha) + \nabla_y \cdot \left(\hat{J}\alpha\hat{\mathbf{F}}^{-1}(\mathbf{v} - \hat{\mathbf{v}}) + \hat{J}\hat{\mathbf{F}}^{-1}\gamma_x\right) - \hat{J}\beta \, d\Omega_y = 0 \,. \tag{4.27}$$

This is the **arbitrary Lagrangian–Eulerian (ALE)** form of a scalar conservation law. The name of it refers to the fact that it interpolates between two more commonly invoked limits, namely, the **Lagrangian description**, in which $\mathbf{y} = \mathbf{X}$, $\hat{\mathbf{v}} = \mathbf{v}$, and $\hat{\mathbf{F}} = \mathbf{F}$, and the **Eulerian description**, in which $\mathbf{y} = \mathbf{x}$, $\hat{\mathbf{F}}$ is identity, and $\hat{\mathbf{v}} = \mathbf{0}$. The Lagrangian description is most common in solid mechanics, while the Eulerian description is most common in fluid dynamics. To bridge this gap in kinematic descriptions, FSI requires an understanding of the ALE description.

Through some further manipulations, the ALE form of a generic balance law can be re-written in a more commonly-seen form that mixes the differential operators ∇_x in space and ∂_r in time. The first step is to change variables in the integration to obtain

$$\int_{\Omega_x} \hat{J}^{-1}\partial_r\left(\hat{J}\alpha\right) + \nabla_x \cdot (\alpha(\mathbf{v} - \hat{\mathbf{v}}) + \gamma_x) - \beta \, d\Omega_x = 0 \,. \tag{4.28}$$

Using the product rule and the identity

$$\partial_r \hat{J} = \hat{J}\nabla_x \cdot \hat{\mathbf{v}} \,, \tag{4.29}$$

(which is analogous to (4.22) when the reference configuration is the material one) we get that the first term in the integrand of (4.28) becomes

$$\partial_r\left(\hat{J}\alpha\right) = \alpha\nabla_x \cdot \hat{\mathbf{v}} + \partial_r\alpha \,. \tag{4.30}$$

Plugging this into (4.28), applying the product rule to $\nabla_x \cdot (\alpha(\mathbf{v} - \hat{\mathbf{v}}))$, and simplifying, we finally arrive at

$$\int_{\Omega_x} \partial_r\alpha + (\mathbf{v} - \hat{\mathbf{v}}) \cdot \nabla_x\alpha + \alpha\nabla_x \cdot \mathbf{v} + \nabla_x \cdot \gamma_x - \beta \, d\Omega_x = 0 \,, \tag{4.31}$$

which is perhaps the mostly widely-used form of the generic ALE balance law.

4.3 Examples

We now apply the kinematics described in Section 4.2, along with a straightforward extension of it developed in Exercise 1 of this chapter, to several common examples of balance laws from continuum mechanics.

4.3.1 Advection–diffusion

If we consider a scalar conserved quantity $\alpha = u$ with source term $\beta = f$ and define the flux to be

$$\gamma_x := -\boldsymbol{\kappa}\nabla_x u \,, \tag{4.32}$$

where the **diffusion tensor** $\boldsymbol{\kappa}$ is a given positive-definite $\mathbb{R}^{d\times d}$-valued field, then we obtain the advection–diffusion equation on a deforming domain:

$$\partial_r u + (\mathbf{v} - \hat{\mathbf{v}}) \cdot \nabla_x u + u\nabla_x \cdot \mathbf{v} - \nabla_x \cdot (\boldsymbol{\kappa}\nabla_x u) = f \,. \tag{4.33}$$

This problem is frequently restricted to the case where the material velocity $\mathbf{v}$ is solenoidal, and the problem simplifies to

$$\partial_r u + \mathbf{a} \cdot \nabla_x u - \nabla_x \cdot (\boldsymbol{\kappa}\nabla_x u) = f \,, \tag{4.34}$$

where $\mathbf{a} := \mathbf{v} - \hat{\mathbf{v}}$ is the **advection velocity**. For different interpretations of u, this equation can be used to model a wide variety of problems, such as heat transfer and particle transport; (4.34) is easily recognizable as the "heat equation" or "diffusion equation" when $\mathbf{a} = \mathbf{0}$.

Remark 31. A common form of the diffusion tensor in isotropic media is $\boldsymbol{\kappa} = \kappa\mathbf{I}$, where $\kappa > 0$ is a scalar diffusion coefficient and $\mathbf{I}$ is the identity tensor. If κ is constant in space, we have the further simplification, $-\nabla_x \cdot (\boldsymbol{\kappa}\nabla_x u) = -\kappa\Delta_x u$, where Δ_x is the spatial Laplacian. Most examples in subsequent chapters will use this simpler form.

Remark 32. The term "convection" is frequently used interchangeably with "advection" in the numerical analysis literature, and (4.34) is sometimes called the "convection–diffusion equation". However, references in applied fields, especially heat transfer, distinguish convection from advection by defining advection to be the transport of some quantity by bulk motion of material, while convection is the combined effects of advection and diffusion. (Following that terminology, "convection–diffusion" would be a redundant term.) For better compatibility with the applied literature, we prefer the term "advection" to describe transport by velocity $\mathbf{a}$.

4.3.2 Mass conservation

In continuum mechanics of both fluids and solids, one typically assumes that no mass flows through the boundary $\partial\Omega_X$ of a material region, and no mass is generated within Ω_X. Thus, considering the Lagrangian case of (4.27), with α equal to the **mass density** ρ, $\gamma_x = \mathbf{0}$, and $\beta = 0$, we get

$$\partial_s(J\rho) = 0 \,. \tag{4.35}$$

This implies that $J\rho$ is a constant at each material point, so we can define

$$\rho_0 = J\rho \quad \Longleftrightarrow \quad \rho = J^{-1}\rho_0 \,, \tag{4.36}$$

where ρ_0 is independent of time and has the interpretation of mass density in the reference configuration. A special case of mass conservation is **incompressibility**, where the mass density never changes:

$$J = 1 \quad \Longleftrightarrow \quad \rho = \rho_0 \quad \Longleftrightarrow \quad \nabla_x \cdot \mathbf{v} = 0 \,. \tag{4.37}$$

The third expression of incompressibility in (4.37) follows from (4.22).

4.3.3 Incompressible Navier–Stokes

The simplest model for low-Mach-number fluid flows is the incompressible Navier–Stokes system, which is obtained from combining a scalar-valued balance law, mass conservation, with a vector-valued one, linear momentum conservation. The typical form of incompressible mass conservation used in this context is the third equation of (4.37),

$$\nabla_x \cdot \mathbf{v} = 0 \,. \tag{4.38}$$

Next, we consider the vector-valued analogue (cf. Exercise 1) of (4.31), with the vector-valued conserved quantity of linear momentum, $\rho\mathbf{v}$, which has the (negative) Cauchy stress $-\boldsymbol{\sigma}$ as its tensor-valued flux $\boldsymbol{\Gamma}_x$ and a body force $\mathbf{f}$ per unit volume as its source term:

$$\rho\left(\partial_r \mathbf{v} + (\mathbf{v} - \hat{\mathbf{v}}) \cdot \nabla_x \mathbf{v}\right) - \nabla_x \cdot \boldsymbol{\sigma} = \mathbf{f} \,, \tag{4.39}$$

where we have used the constraint (4.38) and the fact that $\rho = \rho_0$ is constant to simplify. If one further assumes a Newtonian viscosity model, then the Cauchy stress can be expressed as

$$\boldsymbol{\sigma} = 2\mu \nabla_x^s \mathbf{v} - p\mathbf{I} \,, \tag{4.40}$$

where p is the pressure and $\nabla_x^s \mathbf{v}$ is the symmetrized gradient $\frac{1}{2}(\nabla_x \mathbf{v} + (\nabla_x \mathbf{v})^T)$. Then (given appropriate initial and boundary data), the PDE system given by 4.39 and 4.38 can be solved for the unknown fields $\mathbf{v}$ and p. In this case, the pressure p takes on the mathematical interpretation of a **Lagrange multiplier** field corresponding to the pointwise constraint (4.38); we elaborate much more on this interpretation—and its ramifications for numerical methods—in Section 5.

4.3.4 Hyperelasticity

For modeling solid mechanics, it is most common to use the Lagrangian description, because many applications of solid mechanics entail only mild deformations of material regions, and the constitutive models used for solids often involve state variables associated with material points, the transport of which would become nontrivial in an ALE description with $\hat{\mathbf{v}} \neq \mathbf{0}$. As in our model fluid problem from Section 4.3.3, we consider the simplest class of solid models (without invoking kinematic approximations), namely hyperelastic models. As in Section 4.3.3, we shall combine mass and linear momentum conservation.

In this application, it is most convenient to express mass conservation in the second form given by (4.36),

$$\rho = J^{-1}\rho_0 \,. \tag{4.41}$$

Again following the procedure from Section 4.3.3, we next consider the vector-valued extension of (4.27) (cf. Exercise 1) with linear momentum $\rho\mathbf{v} = J^{-1}\rho_0\mathbf{v}$ as the conserved quantity, (negative) Cauchy stress $-\boldsymbol{\sigma}$ as the flux, and a body force density $\mathbf{f} = J^{-1}\mathbf{f}_0$ as a source term. Standard computational methods for this problem re-write the velocity $\mathbf{v}$ in terms of a displacement field,

$$\mathbf{v} = \partial_s \mathbf{u} \quad \text{where} \quad \mathbf{u}(\mathbf{X}, t) := \boldsymbol{\phi}(\mathbf{X}, t) - \mathbf{X} \quad \Rightarrow \quad \mathbf{F} = \nabla_X \mathbf{u} - \mathbf{I} \,, \tag{4.42}$$

and the vector-valued analogue of (4.27) becomes

$$\rho_0 \partial_s^2 \mathbf{u} - \nabla_X \cdot \mathbf{P} = \mathbf{f}_0 \,, \tag{4.43}$$

where $\mathbf{P}$ is the **first Piola–Kirchhoff stress**,

$$\mathbf{P} := J\boldsymbol{\sigma} \cdot \mathbf{F}^{-T} . \tag{4.44}$$

Hyperelastic constitutive models are those of the form

$$\mathbf{S} := \mathbf{F}^{-1}\mathbf{P} = \frac{\partial \psi}{\partial \mathbf{E}} , \tag{4.45}$$

where ψ is a scalar field giving the density of elastic energy (per unit volume in Ω_X), $\mathbf{S}$ is called the **second Piola–Kirchhoff stress**, and

$$\mathbf{E} = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right) \tag{4.46}$$

is the **Green–Lagrange strain**. Thus, given a potential ψ in terms of $\mathbf{E}$, (4.43) becomes a PDE system for the unknown vector field $\mathbf{u}$.

To better understand what ψ might look like, we shall discuss the Green–Lagrange strain further, then give some examples. Geometrically, $2\mathbf{E}$ can be interpreted as the change in dot product between tangent vectors to material fibers due to a deformation:

$$\mathbf{U} \cdot (2\mathbf{E}) \cdot \mathbf{V} = \mathbf{u} \cdot \mathbf{v} - \mathbf{U} \cdot \mathbf{V} , \tag{4.47}$$

where $\mathbf{U}$ and $\mathbf{V}$ are vector fields on Ω_X and $\mathbf{u} = \mathbf{F}\mathbf{U}$ and $\mathbf{v} = \mathbf{F}\mathbf{V}$ are their pushforwards by ϕ. Changes in dot products provide information about both changes in length (extensional strain) and changes in angle (shear strain). Expanding the definition of $\mathbf{E}$ in terms of displacement,

$$\begin{aligned} 2E_{IJ} &= F_{kI}F_{kJ} - \delta_{IJ} && (4.48)\\ &= (u_{k,I} + \delta_{kI})(u_{k,J} + \delta_{kJ}) - \delta_{IJ} && (4.49)\\ &= u_{J,I} + u_{I,J} + u_{k,I}u_{k,J} , && (4.50) \end{aligned}$$

one can also see that dropping higher-order terms in $\nabla_X\mathbf{u}$ recovers the small strain tensor from linear elasticity, $\boldsymbol{\varepsilon} := \nabla^s_X\mathbf{u}$ (which is the reason for including the seemingly-unwieldy factor of 2 in the geometric definition).

The small-deformation correspondence between $\mathbf{E}$ and the small strain from linear elasticity motivates the formally-simplest hyperelastic model, the **St. Venant–Kirchhoff model**,

$$\psi = \frac{1}{2}\mathbf{E} : C : \mathbf{E} , \tag{4.51}$$

where C is the rank-4 elasticity tensor. However, this model is not stable under compression, and it can be seen that the energy density does not diverge as $J \to 0$, as would be needed to prevent non-physical degenerated deformations. The St. Venant–Kirchhoff model is only well-suited to problems with small deformations (but potentially large rotations, unlike linear elasticity). A slightly more complex model that retains stability under compression is the **compressible neo-Hookean model**,

$$\psi = \frac{1}{2}\left(\mu\left(J^{-2/3}\,\mathrm{tr}\,\mathbf{C} - 3\right) + K\left(\frac{1}{2}\left(J^2 - 1\right) - \ln J\right)\right) , \tag{4.52}$$

where $\mathbf{C} = 2\mathbf{E} + \mathbf{I} = \mathbf{F}^T\mathbf{F}$ is the **right Cauchy–Green tensor** and μ and K are the shear and bulk moduli.

Remark 33. In practice, a variety of different potentials may be referred to as "neo-Hookean" in the literature. The specific form (4.52) is taken from [39].

Remark 34. Incompressible hyperelasticity is commonly used to model materials like rubber or soft biological tissues. As in the case of fluids, this enters as a kinematic constraint, with an associated Lagrange multiplier field. In particular, one has

$$\psi = \psi_{\mathrm{el}} - p(J-1)\,, \tag{4.53}$$

where ψ_{el} is the elastic part of the energy density, depending on $\mathbf{E}$, and p is the Lagrange multiplier for the constraint $J = 1$. Note that the constraint can also be written in terms of $\mathbf{E}$, because

$$J := \det \mathbf{F} = \sqrt{\det \mathbf{C}} = \sqrt{\det(2\mathbf{E} + \mathbf{I})}\,, \tag{4.54}$$

so ψ depends entirely on $\mathbf{E}$ and p. The neo-Hookean model is commonly used for ψ_{el}, simplifying to the standard **incompressible neo-Hookean model**,

$$\psi_{\mathrm{el}} = \frac{\mu}{2}(\operatorname{tr}\mathbf{C} - 3) \tag{4.55}$$

due to the constraint of $J = 1$. In the limit of small deformations, this constraint is approximately $\nabla_X \cdot \mathbf{u} = 0$, and concepts from Chapter 5 can, in practice, be extrapolated to this nonlinear setting.

4.4 Exercises

1. Retrace the manipulations in Section 4.2 of the generic scalar balance law (4.16), but for a generic *vector* balance law of the form

$$\frac{d}{dt}\int_{\Omega_x} \boldsymbol{\alpha}\, d\Omega_x = -\int_{\partial\Omega_x} \boldsymbol{\Gamma}_x \mathbf{n}_x\, d\partial\Omega_x + \int_{\Omega_x} \boldsymbol{\beta}\, d\Omega_x\,, \tag{4.56}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are vector fields and $\boldsymbol{\Gamma}_x$ is a rank-2 tensor-valued flux. Write each step in index notation. *Hint:* Direct notation for each step can be found in [39], with some minor changes in notation and sign conventions.

2. Suppose that the body force density in a hyperelastic problem is a **conservative force field**, of the form $\mathbf{f}_0(\mathbf{X}) = -\nabla_x \psi_{\mathrm{ext}}(\mathbf{X} + \mathbf{u}(\mathbf{X}))$, for some scalar potential ψ_{ext}. Then the kinetic and potential energies of a hyperelastic body with elastic energy density ψ can be expressed as

$$T = \frac{1}{2}\int_{\Omega_X} \rho_0 |\mathbf{v}|^2\, d\Omega_X \quad \text{and} \quad U = \int_{\Omega_X} \psi + \psi_{\mathrm{ext}}\, d\Omega_X\,. \tag{4.57}$$

Define the **Lagrangian**,[1] $L := T - U$, and **action**, $S := \int_{t_1}^{t_2} L\,dt$, on a time interval (t_1, t_2), and show that the principle of stationary action (also known as **Hamilton's principle**), i.e.,

$$D_{\mathbf{w}} S = 0 \quad \forall \mathbf{w} \tag{4.58}$$

(where the Gateaux derivative is taken with respect to displacement), implies the partial differential equation (4.43).

[1]This use of the term "Lagrangian" is distinct from the "Lagrangian description" of a conservation law introduced earlier in this chapter.

3. The following FEniCS script implements the simple St. Venant–Kirchhoff constitutive model mentioned in Section 4.3.4:

```
from dolfin import *
N = 8
mesh = UnitCubeMesh(N,N,N)
k = 1
dx = dx(metadata={"quadrature_degree":2*k})
V = VectorFunctionSpace(mesh,"CG",k)
u = Function(V)
I = Identity(len(u))
def problem(u):
    F = I + grad(u)
    C = F.T*F
    E = 0.5*(C-I)
    K = Constant(1.0e1)
    mu = Constant(1.0e1)
    S = K*tr(E)*I + 2.0*mu*(E - tr(E)*I/3.0)
    psi = 0.5*inner(E,S)
    return F,S,psi
X = SpatialCoordinate(mesh)
u_ex = as_vector(3*[0.1*sin(pi*X[0])
                   *sin(pi*X[1])*sin(pi*X[2]),])

####### Replace #######
f0 = Constant((1,2,3))
#######################

F,S,psi = problem(u)
v = TestFunction(V)
R = derivative(psi*dx,u,v) - inner(f0,v)*dx
J = derivative(R,u)
bc = DirichletBC(V,Constant((0,0,0)),"on_boundary")
solve(R==0,u,[bc,],J=J)
```

Apply the method of manufactured solutions (cf. Exercise 2 in Chapter 3) on a unit cube, with the exact displacement solution $\mathbf{u}_{ex}(\mathbf{X}) = 0.1\sin(\pi X_1)\sin(\pi X_2)\sin(\pi X_3)(\mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3)$. Use the plotting technique from Remark 26 to verify optimal H^1 convergence rates with CG spaces of polynomial degrees 1–3.

4. Use the definition of the Gateaux derivative to manually derive the Jacobian form for the problem from Exercise 3, and implement Newton's method directly, using a Python `for` loop. Verify that the new implementation produces the same solution.

Chapter 5

Coupling between fields: Incompressibility

5.1 Introduction

Our discussion of fluids will be limited to incompressible flows. The pedagogical reasons for this are twofold:

1. We do not have time to discuss the more complicated thermodynamics of compressible flows, and the salient numerical issues faced in hyperbolic problems can be illustrated in the context of stabilizing advection in the incompressible setting.

2. Incompressible flow gives us a case study in both coupling between qualitatively-different fields occupying the same domain (viz., velocity and pressure) and nontrivial satisfaction of inf-sup stability.

This chapter will mostly focus on the issue of inf-sup stability, and various approaches to attaining it in discrete problems. As such, we will study Stokes flow, where the advective term is neglected. Stable treatment of advection will be considered in Chapter 6, although the final means of attaining inf-sup stability that we consider in this chapter (Section 5.4) is closely-related to prominent methods for stabilizing advection, which is why proportionally-greater emphasis is placed on it.

5.2 The Stokes flow model

Following Section 4.3.3, we can derive the Eulerian description of **incompressible Navier–Stokes flow** within the framework of space–time conservation laws from Chapter 4 by choice of the spatial domain as a referential domain (i.e., $\hat{\mathbf{v}} = \mathbf{0}$):

$$\rho \left(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla_x \mathbf{v}\right) - \nabla_x \cdot \boldsymbol{\sigma} = \mathbf{f} \, , \tag{5.1}$$

where we will again assume the Newtonian viscosity model (4.40) for $\boldsymbol{\sigma}$. When the **Reynolds number**, $\mathrm{Re} = \rho|\mathbf{v}|L/\mu$ (where L is some length scale associated with the problem domain), is very small, it may be appropriate to neglect the unsteady and advective terms, to obtain **Stokes flow**:

$$-\nabla_x \cdot \boldsymbol{\sigma} = -\nabla_x \cdot (2\mu\varepsilon(\mathbf{v}) - p\mathbf{I}) = \mathbf{f} \, , \tag{5.2}$$

where we have introduced the notation ε for the symmetric gradient operator. Stokes flow is useful as an illustrative model problem in incompressibility, because it is linear, and thus more easily amenable to rigorous numerical analysis, but it retains the saddle-point structure associated with the Lagrange multiplier and constraint. That structure is what is specifically challenging about incompressibility. (Challenges associated with advection will be addressed in Chapter 6.)

Remark 35. Stokes flow is formally-identical to incompressible linear isotropic elasticity, where μ takes on the role of the shear modulus, and the unknown vector field is interpreted as a displacement, rather than a velocity. Thus, many of the ideas presented here in the context of incompressible fluid flow apply as well to (nearly-)incompressible solid mechanics.

For simplicity, we pose the Stokes problem on a bounded domain $\Omega \subset \mathbb{R}^d$, and impose homogeneous boundary conditions on the velocity $\mathbf{v}$, i.e., $\mathbf{v} = \mathbf{0}$ on $\partial\Omega$. A weak problem corresponding to (5.2) is: Find $(\mathbf{u}, p) \in V \times Q$ such that $\forall(\mathbf{v}, q) \in V \times Q$,

$$a(\mathbf{u}, \mathbf{v}) + b(\mathbf{u}, q) - b(\mathbf{v}, p) = L(\mathbf{v}) \,, \tag{5.3}$$

where $V = \left(H^1_0(\Omega)\right)^d$, $Q = L^2_0(\Omega)$,

$$a(\mathbf{u}, \mathbf{v}) = (2\mu\varepsilon(\mathbf{u}), \nabla\mathbf{v})_{L^2(\Omega)} \,, \tag{5.4}$$

$$b(\mathbf{u}, q) = (\nabla \cdot \mathbf{u}, q)_{L^2(\Omega)} \,, \tag{5.5}$$

$$L(\mathbf{v}) = (\mathbf{f}, \mathbf{v})_{L^2(\Omega)} \,, \tag{5.6}$$

and we have made the change of notation $\mathbf{v} \to \mathbf{u}$ for the velocity solution (and then used $\mathbf{v}$ as a test function), as is more customary in the numerical analysis literature on incompressible flow. (We also drop the subscript "x" from spatial differential operators, as there is no risk of confusion in this setting.) By "$L^2_0(\Omega)$", we mean the subspace of L^2 functions with an average value of zero:

$$L^2_0(\Omega) = \left\{ q \in L^2(\Omega) \text{ s.t. } \int_\Omega q \, d\Omega = 0 \right\} \,. \tag{5.7}$$

It is easy to see that, because $\boldsymbol{\sigma}$ depends only on ∇p, the problem would be ill-posed without some mechanism to pin down the **hydrostatic mode** of constants added to the pressure. The global constraint (5.7) may be difficult to enforce in computations, and an alternative approach in discrete problems is to constrain a single pressure degree of freedom to zero.

Remark 36. The issue of controlling the hydrostatic mode only applies to problems with all Dirichlet boundary conditions, such as the model considered here. If a traction boundary condition is applied to part of the boundary, that will eliminate the hydrostatic mode.

Remark 37. It may not be immediately obvious that a is coercive in $(H^1_0(\Omega))^d$, since it does not at first appear to control the antisymmetric part of the velocity gradient. However, there is a result known as **Korn's inequality**,

$$\|\nabla\mathbf{v}\|_{L^2(\Omega)} \leq C_K \|\varepsilon(\mathbf{v})\|_{L^2(\Omega)} \quad \forall\mathbf{v} \in H^1_0(\Omega) \,, \tag{5.8}$$

which provides H^1_0 coercivity. This is also central to the stability of solid mechanics (which is formally-related to Stokes flow, as noted in Remark 35). Essentially, it says that rotations are bounded by strains (or, in fluid mechanics language, spin is bounded by strain rate) if the displacement (velocity) is fixed at boundaries. The proof of Korn's inequality is more complicated than one might initially expect; an argument using the Fourier transform can be found in [40, Chapter 6, Box 1.1].

Because the test function $(\mathbf{v}, q)$ is arbitrary, we can consider the special cases of $\mathbf{v} = \mathbf{0}$ and $q = 0$ to isolate two coupled **subproblems**:

- Momentum balance: $a(\mathbf{u}, \mathbf{v}) - b(\mathbf{v}, p) = L(\mathbf{v}) \quad \forall \mathbf{v} \in V$,
- Incompressibility: $b(\mathbf{u}, q) = 0 \quad \forall q \in Q$.

Because each involves the solution to the other, these problems cannot be decoupled and solved separately.

The problem (5.3) is a practically-significant example of a non-coercive variational problem. Let us attempt to show coercivity. The full left-hand-side bilinear form is

$$A((\mathbf{u}, p), (\mathbf{v}, q)) = a(\mathbf{u}, \mathbf{v}) + b(\mathbf{u}, q) - b(\mathbf{v}, p) \ . \tag{5.9}$$

Plugging in the same function $(\mathbf{v}, q) \in V \times Q$ to both arguments of A, we get

$$A((\mathbf{v}, q), (\mathbf{v}, q)) = a(\mathbf{v}, \mathbf{v}) \ . \tag{5.10}$$

Even if the form a is coercive over V, A will not be coercive over $V \times Q$, because choosing $(\mathbf{v}, q) = (\mathbf{0}, q)$, we can get $A = 0$ even when the function supplied for both of its arguments is nonzero (and thus necessarily has norm greater than zero).

Problems with the general structure (5.3) are called **saddle-point problems**, due to an analogy to low-dimensional constrained minimization, where the surface on which critical points are sought may look like an equestrian saddle when plotted. For saddle-point problems, A's satisfaction of the conditions of the inf-sup theorem presented earlier are equivalent to the following conditions on a and b:

- Stability of a on a subspace satisfying the constraint equation:

  $$\inf_{\mathbf{u} \in V_0} \sup_{\mathbf{v} \in V_0} \frac{a(\mathbf{u}, \mathbf{v})}{\|\mathbf{u}\|_V \|\mathbf{v}\|_V} \geq C_a > 0 \ , \tag{5.11}$$

  where V_0 is the subspace of V satisfying the constraint:

  $$V_0 = \{\mathbf{v} \in V \ \text{ s.t. } \ b(\mathbf{v}, q) = 0 \ \forall q \in Q\} \ , \tag{5.12}$$

  sometimes referred to as the **kernel** space of the constraint. Frequently, as in the case of Stokes flow, the form a is coercive, and this condition is satisfied in a straightforward way.

- An inf-sup condition on the constraint form, b:

  $$\inf_{q \in Q} \sup_{\mathbf{v} \in V} \frac{b(\mathbf{v}, q)}{\|\mathbf{v}\|_V \|q\|_Q} \geq C_b > 0 \ . \tag{5.13}$$

  By construction, the form b cannot be coercive, since its arguments are in different spaces, so this inf-sup condition is always nontrivial.

These conditions are met by the choices of a, b, V, and Q given above for the weak form of Stokes flow. **However, if we apply the Bubnov–Galerkin method using subspaces $V^h \subset V$ and $Q^h \subset Q$, this may no longer be the case**, as discussed in Chapter 2. Recall that the root of this difficulty is that the supremum over a subset can become smaller than the supremum over the full set. For the optimal convergence analysis given in Section 2.2.1 to apply to Galerkin discretizations of (5.3), we must carefully choose a sequence of spaces such that

$$\inf_{q^h \in Q^h} \sup_{\mathbf{v}^h \in V^h} \frac{b(\mathbf{v}^h, q^h)}{\|\mathbf{v}^h\|_V \|q^h\|_Q} \geq C_b \quad \text{as the refinement parameter } h \to 0\ , \tag{5.14}$$

where $C_b > 0$ is independent of h. We point out now the general trend that lowering the dimension of Q^h relative to V^h makes discretizations more stable, but enforces the constraint less accurately, because there are fewer constraint equations in the discrete system. This trend can be expected from the form of the inf-sup condition on b: because the infimum is taken over Q^h, that is the space that may increase the left-hand side by shrinking, while the supremum is taken over V^h, so that is the space that may increase the left-hand side by growing.

Remark 38. The saddle-point form of the inf-sup condition (5.13) is sometimes called the **Brezzi form**, as this was the setting considered by F. Brezzi in [41]. The form given in (1.41) is the **Babuška form**, which was shown by I. M. Babuška in [42]. The proof of equivalence is spelled out thoroughly in the excellent notes entitled "Babuška $\iff$ Brezzi?", by L. F. Demkowicz [17].

Remark 39. Inf-sup stability of pairs of pressure and velocity spaces can be seen as a generalization of the logic used to justify the "staggered grid" finite difference/volume methods prevalent in computational fluid dynamics. Recall the exposition from Sections 2.5.1 and 2.5.2 on the equivalence of certain finite difference/volume methods with finite element methods; similar arguments apply to Galerkin discretizations of incompressibility on structured finite element meshes.

Remark 40. While proofs of inf-sup stability may be quite technical, it is possible to test the inf-sup condition *numerically*, by solving eigenvalue problems. See, e.g., [43].

5.3 Some stable elements for Stokes flow

A vast literature on stable elements for incompressible flow exists, as reviewed by Boffi et al. [44]. In this section, we simply highlight and compare a few examples of stable pressure–velocity pairs.

5.3.1 Taylor–Hood elements

Perhaps the most well-known family of stable elements for incompressible flow are the **Taylor–Hood** family of elements. These are given by $V^h = (\mathrm{CG}^h_k)^d$ and $Q^h = \mathrm{CG}^h_{k-1}$, for $k \geq 2$. A simple FEniCS implementation of the method of manufactured solutions (cf. Exercise 2 in Chapter 3), using the Taylor–Hood element, is as follows:

```
from dolfin import *
N = 64
k = 2
```

```
mu = Constant(1.0)
mesh = UnitSquareMesh(N,N)
x = SpatialCoordinate(mesh)
I = Identity(mesh.geometry().dim())
cell = mesh.ufl_cell()
Ve = VectorElement("CG",cell,k)
Qe = FiniteElement("CG",cell,k-1)
W = FunctionSpace(mesh,MixedElement([Ve,Qe]))

# Take the curl of a potential for solution
# to manufacture:
u3 = (sin(pi*x[0])*sin(pi*x[1]))**2
u_exact = as_vector([u3.dx(1),-u3.dx(0)])
p_exact = sin(2.0*pi*x[0])*sin(3.0*pi*x[1])
def sigma(u,p):
    return 2.0*mu*sym(grad(u)) - p*I
f = -div(sigma(u_exact,p_exact))

# u_exact is zero on boundaries.
bc = DirichletBC(W.sub(0),Constant((0,0)),
                 "on_boundary")

# Galerkin formulation:
u,p = TrialFunctions(W)
v,q = TestFunctions(W)
res = (inner(sigma(u,p),grad(v)) + div(u)*q
       - dot(f,v))*dx
up = Function(W)
solve(lhs(res)==rhs(res),up,bc)

# Check H1 error; can verify optimal (k-th) order
# by modifying parameter N at top of script.
u,p = split(up)
grad_e = grad(u-u_exact)
print(sqrt(assemble(inner(grad_e,grad_e)*dx)))
```

Note that, in the above code example, the pressure is unconstrained. In computations, it is frequently the case that a given linear solver will end up picking the constant up to which p^h is defined in an ad hoc way. If one wants to compare p^h with an exact solution, the free constant must be pinned down, e.g., by applying a boundary condition on p at a corner of the domain.

Remark 41. FEniCS also implements $\mathbb{R}$ as function space, viz. `FunctionSpace(mesh,"R",0)`, which can be included in the mixed space for a Lagrange multiplier to enforce the average pressure constraint, but this is not particularly efficient, since a single degree-of-freedom is coupled to the entire domain.

5.3.2 Scott–Vogelius elements

Recalling the de Rham complex from Section 2.4.4, we might consider $V^h = (\mathrm{CG}^h_k)^d$ and ask: What space does the divergence of a function from V^h lie in? This motivates the **Scott–Vogelius** family of elements, with $Q^h = \mathrm{DG}^h_{k-1}$. Clearly, Q^h contains the divergence of every vector field in V^h. Thus, if the constraint equation

$$b(\mathbf{u}^h, q^h) = 0 \tag{5.15}$$

is satisfied $\forall q^h \in Q^h$, it is satisfied for the specific choice $q^h = \nabla \cdot \mathbf{u}^h$. This then implies that $\nabla \cdot \mathbf{u}^h$ is zero almost everywhere. That is a remarkable result: weak incompressibility implies *strong* incompressibility, in the discrete setting. However, this accuracy of enforcing $\nabla \cdot \mathbf{u}^h = 0$ comes at the cost of stability. Notice that DG^h_{k-1} is a larger space than the CG^h_{k-1} space used in the Taylor–Hood discretization. As such, we might expect to have problems ensuring the inf-sup stability of the Scott–Vogelius element, and stability results are, at present, only available for the 2D case, with high k and awkward restrictions on mesh generation. Determining sufficient conditions for stability in 3D remains an open problem.

The FEniCS script given in Section 5.3 for Taylor–Hood can be easily modified to implement the Scott–Vogelius element, by changing the definition of `Qe` to a DG space. However, the instability of lower-order Scott–Vogelius elements becomes apparent, in the large errors obtained without also choosing $k \geq 4$, as expected from the theoretical results about this family of elements.

5.3.3 Divergence-conforming B-splines

Despite the Scott–Vogelius element's remarkable pointwise conservation properties, the rather limited set of scenarios in which it is stable has relegated it to a status of relative obscurity outside of the applied math community. More recently, though, an H^1-conforming (or even more regular) generalization of the Raviart–Thomas element has emerged in isogeometric analysis.

The gist of the construction is as follows. In 1D, a family of polynomial spline functions, called "B-splines", has the property that

$$\frac{d}{d\xi} : \widehat{\mathcal{S}}^k_\alpha \to \widehat{\mathcal{S}}^{k-1}_{\alpha-1} , \tag{5.16}$$

where $\mathcal{S}^k_\alpha$ is the space of spline functions with polynomial degree k and inter-element continuity (i.e., number of continuous derivatives) α. One takes tensor products of these spline spaces to obtain B-splines defined on higher-dimensional "patches", e.g., the space $\widehat{\mathcal{S}}^{k_1,k_2}_{\alpha_1,\alpha_2}$ of functions on a 2D rectangular domain would be the space of tensor products of functions in $\widehat{\mathcal{S}}^{k_1}_{\alpha_1}$ and $\widehat{\mathcal{S}}^{k_2}_{\alpha_2}$ on 1D intervals. If one then defines a vector-valued space where each component is in one of these scalar-valued tensor product spaces, one can form discrete de Rham complexes. For example, if we define

$$\widehat{\mathcal{RT}} := \widehat{\mathcal{S}}^{k_1+1,k_2}_{\alpha_1+1,\alpha_2} \times \widehat{\mathcal{S}}^{k_1,k_2+1}_{\alpha_1,\alpha_2+1} , \tag{5.17}$$

Then it is easy to see that

$$\widehat{\nabla} \cdot \widehat{\mathcal{RT}} \subset \widehat{\mathcal{S}}^{k_1,k_2}_{\alpha_1,\alpha_2} =: \widehat{\mathcal{W}} , \tag{5.18}$$

where $\widehat{\nabla}\cdot$ is the divergence operator on the parametric space on which the spline functions are defined. In other words, it becomes a higher-regularity generalization of the Raviart–Thomas element, which, for $\alpha_1, \alpha_2 \geq 0$ is not just $H(\mathrm{div})$- but H^1-conforming. The construction above applies to splines defined on rectangular parametric grids, but, if these functions are pushed forward by the Piola transform, we get, from the Piola identity (4.10), that

$$\nabla \cdot \mathcal{RT} \subset \mathcal{W} . \tag{5.19}$$

The pushed-forward spaces $\mathcal{RT}$ and $\mathcal{W}$ are defined as follows. If the parametric domain is pushed forward by $\phi : \widehat{\Omega} \to \Omega$, then $\widehat{\mathbf{u}} \in \widehat{\mathcal{RT}}$ is pushed forward to $\mathbf{u} \in \mathcal{RT}$ like

$$\mathbf{u} = \frac{1}{J}\mathbf{F}\widehat{\mathbf{u}} , \tag{5.20}$$

where $\mathbf{F}$ and J are the deformation gradient and Jacobian determinant associated with the deformation ϕ, and $\widehat{p} \in \widehat{\mathcal{W}}$ is pushed forward to $p \in \mathcal{W}$ like

$$p = \frac{1}{J}\widehat{p} . \tag{5.21}$$

Clearly, if we use $\mathcal{RT}$ to discretize velocity and $\mathcal{W}$ to discretize pressure in Galerkin's method for Stokes flow, we obtain an exactly solenoidal velocity field, under the assumption that these spaces are inf-sup stable. This stability result is proven without overly-restrictive assumptions in [45], and the exact mass conservation is shown to be beneficial for complicated 3D problems in [46, 47, 48].

Isogeometric spaces are not natively-supported in FEniCS, but the library tIGAr [37, 49] can be used to compute with them in the FEniCS framework. The tIGAr library includes functionality for divergence-conforming B-spline discretizations, as illustrated in some of the demos available in its repository.

5.4 The pressure-stabilizing Petrov–Galerkin method

We can in fact avoid the difficulty of identifying inf-sup stable pressure–velocity pairs by looking beyond the Bubnov–Galerkin method to coercive Petrov–Galerkin methods that will attain optimal accuracy for arbitrary choices of V^h and Q^h. Another benefit of this approach is that the resulting algebraic equations are also no longer saddle-point systems; this makes them much easier to solve approximately with off-the-shelf implementations of Krylov subspace methods, which are essentially mandatory for large 3D problems.

The most well-known of these stabilized formulations is the **pressure-stabilizing Petrov–Galerkin (PSPG)** method. The idea is to add a term to Galerkin's method which weights the residual of the strong momentum balance problem against ∇q^h, to obtain a $\nabla p^h \cdot \nabla q^h$ term that is coercive in H^1; the zero-average constraint on L^2_0 implies that a Poincaré-like inequality (cf. (1.30)) holds for $Q^h \subset H^1(\Omega) \cap L^2_0(\Omega)$, so this term's control over the H^1 seminorm is sufficient. However, there are two difficulties that arise in formulating such a method:

1. Weighting the strong momentum residual by ∇q^h leads also to a term $-\nabla \cdot \left(2\mu\varepsilon\left(\mathbf{u}^h\right)\right) \cdot \nabla q^h$, which we have no control over the coercivity of; the stabilizing term must be scaled with a carefully-selected parameter so that we can "hide" this term behind something coercive.

2. The resulting formulation is not uniformly-bounded in the norm in which it is coercive, so we need to follow a non-standard error analysis to prove optimal convergence.

To be more precise, the PSPG formulation is as follows: Find $(\mathbf{u}^h, p^h) \in V^h \times Q^h$ such that $\forall(\mathbf{v}^h, q^h) \in V^h \times Q^h$,

$$a(\mathbf{u}^h, \mathbf{v}^h) + b(\mathbf{u}^h, q^h) - b(\mathbf{v}^h, p^h) - L(\mathbf{v}^h) + \left(-\nabla \cdot \boldsymbol{\sigma}(\mathbf{u}^h, p^h) - \mathbf{f}\,, \tau_{\mathrm{M}} \nabla q^h\right)_{\Omega'} = 0\,, \tag{5.22}$$

where $\tau_{\mathrm{M}} \geq 0$ is the **stabilization parameter** and the notation $(\cdot,\cdot)_{\Omega'}$ refers to an L^2 inner product restricted to element interiors,[1] i.e.,

$$(u, v)_{\Omega'} = \sum_{e=1}^{N_{\mathrm{el}}} (u, v)_{L^2(\Omega_e)}\,. \tag{5.23}$$

The parameter τ_{M} must be selected carefully, in a mesh-dependent way, to ensure optimal stability and consistency. For stability analysis, we isolate the bilinear form from the linear problem residual (5.22),

$$A_{\mathrm{stab}}((\mathbf{u}, p), (\mathbf{v}, q)) = a(\mathbf{u}, \mathbf{v}) + b(\mathbf{u}, q) - b(\mathbf{v}, p) + (-\nabla \cdot (2\mu\varepsilon(\mathbf{u})) + \nabla p, \tau_{\mathrm{M}} \nabla q)_{\Omega'}\ , \tag{5.24}$$

and consider its coercivity in the norm $|||\cdot|||$ on $V^h \times Q^h$, defined by

$$2|||(\mathbf{u}, p)|||^2 = 2\mu\|\varepsilon(\mathbf{u})\|_{L^2}^2 + \tau_{\mathrm{M}}|q|_{H^1}^2\,. \tag{5.25}$$

We get

$$A_{\mathrm{stab}}((\mathbf{v}, q), (\mathbf{v}, q)) = 2|||(\mathbf{v}, q)|||^2 + (-\nabla \cdot (2\mu\varepsilon(\mathbf{v})), \tau_{\mathrm{M}} \nabla q)_{\Omega'}\ . \tag{5.26}$$

If we apply the Cauchy–Schwartz and Young's inequalities to the last term of (5.26), we get the upper bound

$$(-\nabla \cdot (2\mu\varepsilon(\mathbf{v})), \tau_{\mathrm{M}} \nabla q)_{\Omega'} \leq \frac{\epsilon\mu^2}{2}\|\nabla \cdot (2\varepsilon(\mathbf{v}))\|_{\Omega'}^2 + \frac{\tau_{\mathrm{M}}^2}{2\epsilon}|q|_{H^1}^2 \quad \forall \epsilon > 0\,. \tag{5.27}$$

On a quasi-uniform mesh of element size h, for standard CG spaces, we have an inverse estimate (cf. (2.34)):

$$\|\nabla \cdot (2\varepsilon(\mathbf{v}))\|_{L^2(\Omega^e)}^2 \leq 2C_{\mathrm{inv}} h^{-2}\|\varepsilon(\mathbf{v})\|_{L^2(\Omega^e)}^2 \quad \forall \mathbf{v} \in V^h\,. \tag{5.28}$$

Choosing $\epsilon = \tau_{\mathrm{M}}$, then

$$A_{\mathrm{stab}}((\mathbf{v}, q), (\mathbf{v}, q)) \geq 2\mu\left(1 - \frac{1}{2}\tau_{\mathrm{M}} C_{\mathrm{inv}}\mu h^{-2}\right)\|\varepsilon(\mathbf{v})\|_{L^2(\Omega)}^2 + \frac{\tau_{\mathrm{M}}}{2}|q|_{H^1}^2\,. \tag{5.29}$$

Thus, if we select

$$\tau_{\mathrm{M}} \leq \frac{h^2}{C_{\mathrm{inv}}\mu}\,, \tag{5.30}$$

we have coercivity of A_{stab} in the norm $|||\cdot|||$, with at least unit constant.

The next question is whether stability in this mesh-dependent norm is sufficient for optimal convergence. Because the pressure contribution to $|||\cdot|||$ is scaled by τ_{M} and goes to zero under refinement, the b terms of A_{stab} are not uniformly bounded in this norm. Thus the question is not

[1]This is necessary because $\nabla \cdot \boldsymbol{\sigma}$ involves second derivatives of $\mathbf{u}^h$; for $\mathbf{u}^h$ in a CG finite element space, $\nabla\nabla\mathbf{u}^h$ includes singular Dirac layers at interior mesh facets, which are not square integrable.

trivial. An outline of the proof is as follows. As usual, we begin with an error decomposition into method and interpolation components, i.e.,

$$e = (\mathbf{e}_u, e_p) = (\mathbf{u} - \mathbf{u}^h, p - p^h) = (\mathbf{e}_u^h + \boldsymbol{\eta}_u, e_p^h + \eta_p) = e^h + \eta \,, \tag{5.31}$$

where the method error e^h is in the discrete space $V^h \times Q^h$. As in the standard inf-sup error analysis, we seek to bound these method errors in terms of interpolation error, η. Assume that τ_{M} satisfies the bound (5.30) with equality. Then

$$\left|\left|\left|e^h\right|\right|\right|^2 \leq A_{\mathrm{stab}}\left(e^h, e^h\right) \tag{5.32}$$

$$= A_{\mathrm{stab}}\left(-\eta, e^h\right) \,, \tag{5.33}$$

where we have use coercivity followed by strong consistency. The remainder of the derivation proceeds by expanding A_{stab} and applying Young's inequality to every term, with judiciously-chosen ϵs, so that everything involving e^h can be moved to the left-hand side to "hide behind" half of $|||e|||^2$, in a manner quite similar to the application of Young's inequality to show coercivity. The remaining terms on the right-hand side in terms of $(\boldsymbol{\eta}_u, \eta_p)$ are of optimal order in h, assuming that the interpolation error is chosen as the best possible in a standard CG$_k$ space. The details of this procedure are left to the reader as Exercise 1.

Remark 42. In practice, PSPG is frequently combined with an additional L^2 penalty on $\nabla \cdot \mathbf{u}^h$, which manifests as adding

$$+ \left(\tau_{\mathrm{C}} \nabla \cdot \mathbf{u}^h, \nabla \cdot \mathbf{v}^h\right)_{L^2} \tag{5.34}$$

to A_{stab}, where τ_{C} is a second stabilization parameter. This is referred to as **least squares on the incompressibility constraint (LSIC)** in the engineering literature and as **grad-div stabilization** in the applied math literature. The logic behind the latter name is to consider the corresponding strong form of the penalty term. The optimal choice of the LSIC stabilization parameter is $\tau_{\mathrm{C}} \sim h^2/\tau_{\mathrm{M}}$. LSIC/grad-div stabilization is frequently also added to Galerkin discretizations using inf-sup stable pressure–velocity pairs.

Considering the same benchmark problem as used for the Taylor–Hood element in Section 5.3, the PSPG method (augmented with LSIC/grad-div stabilization) can be implemented in FEniCS as follows:

```
from dolfin import *

# Whether or not to include stabilization:
USE_STAB = True

N = 32
k = 1
mu = Constant(1.0)
mesh = UnitSquareMesh(N,N)
x = SpatialCoordinate(mesh)
I = Identity(mesh.geometry().dim())
cell = mesh.ufl_cell()
```

```
Ve = VectorElement("CG",cell,k)
Qe = FiniteElement("CG",cell,k)
W = FunctionSpace(mesh,MixedElement([Ve,Qe]))

# Take the curl of a potential for solution
# to manufacture:
u3 = (sin(pi*x[0])*sin(pi*x[1]))**2
u_exact = as_vector([u3.dx(1),-u3.dx(0)])
p_exact = sin(2.0*pi*x[0])*sin(3.0*pi*x[1])
def sigma(u,p):
    return 2.0*mu*sym(grad(u)) - p*I
def strongRes(u,p,f):
    return -div(sigma(u,p)) - f
f = strongRes(u_exact,p_exact,Constant((0,0)))

# u_exact is zero on boundaries.
bc = DirichletBC(W.sub(0),Constant((0,0)),
                 "on_boundary")

# Galerkin formulation:
u,p = TrialFunctions(W)
v,q = TestFunctions(W)
resGalerkin = (inner(sigma(u,p),grad(v)) + div(u)*q
               - dot(f,v))*dx

# Stabilization:
Cinv = Constant(1e0*k**2)
h = CellDiameter(mesh)
tau_M = h*h/(Cinv*mu)
resPSPG = tau_M*inner(strongRes(u,p,f),grad(q))*dx
tau_C = h*h/tau_M
resLSIC = tau_C*div(u)*div(v)*dx
if(USE_STAB):
    resStab = resPSPG + resLSIC
else:
    # Necessary to avoid singularity of problem
    # and NaN pressures without PSPG.
    resStab = Constant(DOLFIN_EPS)*p*q*dx

# Formulation:
res = resGalerkin + resStab

up = Function(W)
solve(lhs(res)==rhs(res),up,bc)
```

```
# Check H1 error; can verify optimal (k-th) order
# by modifying parameter N at top of script.
u,p = split(up)
grad_e = grad(u-u_exact)
print(sqrt(assemble(inner(grad_e,grad_e)*dx)))

# Visualize pressure; with USE_STAB==False,
# this shows a checkerboard mode.
from matplotlib import pyplot as plt
plot(p)
plt.show()
```

The selection of "C_{inv}" in the above code is made on an ad hoc basis, but well-designed stabilized formulations are typically insensitive to the exact values of such constants. If the variable `USE_STAB` is modified to be `False`, then PSPG and LSIC are not used, and the pressure field exhibits extreme oscillatory behavior. In classical finite difference methods for incompressible flow, this is called a "checkerboard mode", because of its visual appearance when plotted on a structured finite difference grid, where adjacent grid points alternate between two extreme colors.

Remark 43. Instead of relying on "C_{inv}" and "h", it is possible to get sharp inverse estimates by solving a small eigenvalue problem in each element of the mesh, as elaborated in [50]; this would completely eliminate the "parameter dependence", that is sometimes decried as a disadvantage of stabilized methods, although, in practice, fixed values have been found effective across diverse flow regimes and application domains.

5.5 Exercises

1. Assume that the PSPG formulation of Section 5.4 is applied using the same CG_k space for the pressure and each component of the velocity. (This is called **equal-order interpolation** in the literature.) Carry out the analysis sketched at the end of Section 5.4 to obtain $|||e||| \leq Ch^k$, with C independent of h, assuming that the exact solution $(\mathbf{u}, p)$ is sufficiently smooth.

   Hint: The only extra "trick" needed beyond what is described in the proof sketch given in Section 5.4 is to notice that

   $$\left(e_p^h, \nabla \cdot \boldsymbol{\eta}_u\right)_{L^2} = -\left(\nabla e_p^h, \boldsymbol{\eta}_u\right)_{L^2} , \tag{5.35}$$

   where the boundary term from this integration-by-parts can be assumed to go to zero.

   Hint: If stuck, feel free to consult the original reference [51].

Remark 44. The proof techniques of using a mesh-dependent norm and "hiding" terms via Young's inequality and inverse inequalities are recurring themes in the analysis of stabilized Petrov–Galerkin methods, and essential also to following upcoming analysis of stabilized advection in Chapter 6.

2. Adapt the hyperelastic code example of Exercise 3 from Chapter 4 to use the incompressible neo-Hookean model mentioned in Remark 34. Use a Taylor–Hood discretization of displacement and Lagrange multiplier. Check the convergence rate of the displacement field under mesh refinement.

   Hint: To check accuracy with the method of manufactured solutions, the exact solution's displacement field needs to satisfy the incompressibility constraint. Note that, for geometrically-nonlinear hyperelasticity, $J = 1$ is *not* equivalent to the displacement $\mathbf{u}$ being solenoidal.

Chapter 6

Coupling between scales: Variational multiscale modeling

6.1 Introduction

We have already seen, in Chapter 2, how the standard error bound for Galerkin's method becomes unhelpful for **advection–diffusion** problems of the form derived in Section 4.3.1: Given $\kappa > 0$, $\mathbf{a} : \Omega \to \mathbb{R}^d$ with $\nabla \cdot \mathbf{a} = 0$, and $f : \Omega \to \mathbb{R}$, find $u : \Omega \to \mathbb{R}$ such that

$$-\nabla \cdot (\kappa \nabla u) + \mathbf{a} \cdot \nabla u = f \ , \tag{6.1}$$

where we may have $\kappa \ll |\mathbf{a}|L$, for length scale L. In this chapter, we relate the breakdown of Galerkin's method when $L = h$ to the fact that exact solutions to (6.1) can contain very small-scale features, such as boundary layers (or, in the related Navier–Stokes problem, turbulent eddies), which may not be practically-resolvable with finite element meshes. The coupling of these unresolved scales to the discrete solution must be accounted for in some rational way to obtain an effective numerical method. This viewpoint and the numerical methods emanating from it collectively fall under the label **variational multiscale (VMS)**. Interestingly, Petrov–Galerkin stabilized methods, such as the PSPG method for Stokes flow discussed in Section 5.4, pre-date the VMS concept, but can be formally derived from VMS reasoning, albeit with some ad hoc approximations.

Strongly-consistent Petrov–Galerkin stabilized methods were originally developed by T. J. R. Hughes and his group, during the 1970s and 1980s, with [52] being among the most highly-cited papers of all time on the topic of finite element analysis. During this era, stabilized formulations were developed essentially by guess-and-check, guided perhaps by some loose intuition about what formulations may turn out to have desirable properties. However, there are endlessly-many strongly-consistent variational formulations corresponding to a given PDE system, and, as should be clear from the exercise on PSPG from Chapter 5, rigorously proving convergence takes some work, even for the simplest problems. Guessing formulations and checking their properties is therefore not an efficient process of discovery.

The concept of VMS was developed in the 1990s and 2000s, also by Hughes and collaborators [53], as a means of providing a framework to devise effective stabilized methods for new problems, or to improve on existing stabilized methods. However, the application of VMS to derive stabilized methods still typically involves some "leap of faith", in which a non-rigorous approximation is substituted for the solution to an infinite-dimensional problem, and the formulations that emerge from VMS derivations must still be verified using the same analytical tools needed for ad hoc stabilized methods.

The present chapter provides a brief overview of the VMS concept in Section 6.2. Section 6.3 then applies VMS ideas to derive a popular stabilized formulation for advection–diffusion. Section 6.5 reintroduces the PSPG method from Section 5.4 within the VMS framework, and Section 6.6 combines VMS-based stabilization of advection and incompressibility to obtain a stabilized formulation of Navier–Stokes flow, which doubles as an implicit large-eddy simulation filter for turbulence modeling.

6.2 The VMS concept

As noted in Section 6.1 the role of VMS analysis in development of practical numerical methods is largely *formal*, in the sense that it is used to inspire the form of stabilized methods, which must then be analyzed by other means. As such, our presentation of it in this section takes some liberties with functional analysis for the sake of expediting the derivation of stabilized methods, which is our purpose in presenting this material. The main abuse of notation we adopt is to conflate duality pairing with the L^2 inner product, in a manner analogous to the common notation

$$\langle \delta_x, \phi \rangle = \int \delta(x-y)\phi(y)\, dy = \phi(x) \tag{6.2}$$

seen in the scientific literature and mentioned in Chapter 1. We will act as if objects such as $-\Delta u$ with $u \in H^1$ are in L^2, even though they may be singular distributions. A more rigorous account of the abstract VMS theory in terms of duality pairing may be found in [54, Section 2], for which this class' introduction to functional analysis should provide sufficient background.

With that notational disclaimer out of the way, we state an abstract linear partial differential equation (PDE) problem on some domain $\Omega \subset \mathbb{R}^d$ as: Find u such that

$$\begin{cases} \mathcal{L}u(x) = f(x) & x \in \Omega\,, \\ u(x) = 0 & x \in \partial\Omega\,, \end{cases} \tag{6.3}$$

where $\mathcal{L}$ is some differential operator (e.g., $\mathcal{L} = -\kappa\Delta + \mathbf{a}\cdot\nabla$ for the advection–diffusion problem) and f is a given source term. The corresponding weak problem is: Find $u \in V$ such that $\forall v \in V$,

$$a(u,v) = L(v)\,, \tag{6.4}$$

where

$$a(u,v) = (\mathcal{L}u, v) \quad \text{and} \quad L(v) = (f,v)\,, \tag{6.5}$$

V is the infinite-dimensional space over which the weak problem is posed (e.g., H^1 in the case of advection–diffusion), and

$$(u,v) = \int_\Omega uv\, d\Omega \tag{6.6}$$

is the L^2 inner product, albeit understood in the formal sense discussed at the beginning of the section, which, for full rigor, would need to be translated into the language of distributions and duality pairing. We also define, for later use, the **formal L^2 adjoint** (abbreviated, in appropriate context, to "adjoint") of $\mathcal{L}$, denoted $\mathcal{L}^*$. It is defined by the property

$$a(u, v) = (\mathcal{L}u, v) = (u, \mathcal{L}^* v) \quad \forall v \in V \ . \tag{6.7}$$

We now introduce a **scale separation** in the weak problem, by representing the space V as a direct sum of two subspaces:

$$V = V^h \oplus V' \ , \tag{6.8}$$

i.e., every $v \in V$ has a unique representation

$$v = v^h + v' \ , \tag{6.9}$$

where $v^h \in V^h$ and $v' \in V'$. As suggested by the notation, V^h will take on the interpretation of a finite-dimensional discrete or **coarse-scale space**, while V' is the infinite-dimensional **fine-scale space**. The decomposition of V into coarse- and fine-scale components is non-unique. Even for a fixed V^h, the corresponding V' depends on the choice of **projector**, $\mathbb{P}^h : V \to V^h$, which we shall return to momentarily.

With this notation, the weak problem (6.4) for u can be re-written as a coupled problem for the coarse- and fine-scale components of u: Find $(u^h, u') \in V^h \times V'$ such that $\forall (v^h, v') \in V^h \times V'$

$$a\left(u^h + u', v^h + v'\right) = L\left(v^h + v'\right) \ , \tag{6.10}$$

or, isolating the coupled subproblems via linearity,

$$a\left(u^h, v^h\right) = L\left(v^h\right) - a\left(u', v^h\right) \quad \forall v^h \in V^h \ , \tag{6.11}$$

$$a\left(u', v'\right) = L\left(v'\right) - a\left(u^h, v'\right) \quad \forall v' \in V' \ . \tag{6.12}$$

The claim is that the "perfect" numerical method is one that can solve the finite system of equations implied by (6.11) to obtain

$$u^h = \mathbb{P}^h u \ , \tag{6.13}$$

for a projector $\mathbb{P}^h$ of our choosing. However, (6.11) is driven by u', which is the solution to the infinite-dimensional problem (6.12).

A special case in which we can still solve exactly for u^h is the following: a is an inner product, and $\mathbb{P}^h$ is orthogonal projection onto V^h with respect to a. Then $a(u', v^h) = 0$, and we can simply solve $a(u^h, v^h) = L(v^h) \ \forall v^h \in V^h$, i.e., the Bubnov–Galerkin method. This provides some insight into why that method is so effective in symmetric positive definite problems whose energy norms are equivalent, with moderate constants, to norms in which we consider approximation meaningful (e.g., isotropic steady heat conduction, whose bilinear form is a constant times the H^1_0 inner product). However, in many cases of practical interest, those conditions are not satisfied. The problem may be non-symmetric (e.g., advection–diffusion) or we may want a good approximation in one topology, while the a-norm is bounded above and below by the desired norm with wildly different constants (e.g., nearly-incompressible elasticity). In such cases, we must look more closely at the scale-separated problem to devise effective numerical formulations.

A first observation is that the fine-scale problem (6.12) is driven by the coarse-scale strong PDE residual:

$$a(u', v') = -\left(a\left(u^h, v'\right) - L(v')\right) \tag{6.14}$$

$$= -\left(\mathcal{L}u^h - f, v'\right) \tag{6.15}$$

$$(\mathcal{L}u', v') = \left(-r^h, v'\right) \ , \tag{6.16}$$

where $r^h := \mathcal{L}u^h - f$. Let us introduce a **fine-scale Green's operator**, $\mathcal{G}'$, which serves as a solution operator to the fine-scale subproblem, i.e.,

$$u' = -\mathcal{G}' r^h \ . \tag{6.17}$$

We can represent the action of this operator as convolution with a **fine-scale Green's function**, g', such that

$$u'(y) = \left(-\mathcal{G}' r^h\right)(y) = -\int_\Omega g'(x, y) r^h(x)\, dx \ , \tag{6.18}$$

where $g(\cdot, y)$ is the solution to

$$(\mathcal{L}^* g'(\cdot, y), v') = \left(\delta_y, v'\right) \quad \forall v' \in V' \ . \tag{6.19}$$

(If this is unclear, it can be verified by plugging $g'(\cdot, y)$ into (6.16) as v'.) Assuming, for a moment, that $\mathcal{G}'$ (or its representation g') may somehow be approximated, then the coarse-scale subproblem (6.11) becomes

$$a\left(u^h, v^h\right) + a\left(-\mathcal{G}' r^h, v^h\right) = L\left(v^h\right) \quad \forall v^h \in V^h \ , \tag{6.20}$$

or,

$$a\left(u^h, v^h\right) + \left(\mathcal{G}' r^h, -\mathcal{L}^* v^h\right) = L\left(v^h\right) \quad \forall v^h \in V^h \ , \tag{6.21}$$

where r^h depends only on the coarse-scale solution u^h. Of course the actual problem for g', (6.19), is infinite-dimensional, and not solvable exactly, outside of trivial 1D examples. Thus, the design of a VMS-based stabilized method hinges on approximating the action of $\mathcal{G}'$ in some computationally-cheap way.

What would be an appropriate approximation of $\mathcal{G}'$? In general, this depends on the desired projector, $\mathbb{P}^h$. For many problems, H^1_0 optimality of u^h is considered to be a "good approximation" of the exact solution. Proceeding with that projector, consider the intuition we get from the 1D. Recall from Exercise 6 in Section 2.7 that H^1_0 projection to CG finite element spaces in 1D is exact at element boundaries. From this, we can easily infer that the space V' consists of functions going to zero at element boundaries, in which case we can split the problem (6.19) up element-by-element, and see that $g'(\cdot, y)$ is entirely localized to the element containing y. Thus, in problems where we desire H^1_0 optimality of u^h, we might consider a **localized approximation** of the form

$$u'(y) = -\int_\Omega g'(x, y) r^h(x)\, dx \approx -\tau r^h(y) \ , \tag{6.22}$$

where $\tau \in \mathbb{R}$ is called a **stabilization parameter**, for reasons which will be clear soon. While the complete localization that occurs in 1D no longer holds in higher dimensions, it is indeed the case that the H^1_0 projector leads to better localization of g' than, e.g., L^2, as illustrated in [54, Figures 3.9–3.10]. We now proceed to several illuminating examples in the subsequent sections, where we use such a localized approximation of $\mathcal{G}'$ to derive stabilized formulations.

6.3 VMS stabilization of advection

The development of stabilized Petrov–Galerkin methods began with the quest for simultaneously stable and accurate methods of approximating advective transport. Let us consider the problem (6.1), which, in weak form, with zero Dirichlet boundary conditions, is: Find $u \in H_0^1(\Omega)$ such that $\forall v \in H_0^1(\Omega)$,

$$(\kappa\nabla u, \nabla v) + (\mathbf{a} \cdot \nabla u, v) = (f, v) \ . \tag{6.23}$$

In the notation for our abstract statement of VMS, we have

$$\mathcal{L} = -\kappa\Delta + \mathbf{a} \cdot \nabla \ , \tag{6.24}$$

which implies

$$\mathcal{L}^* = -\kappa\Delta - \mathbf{a} \cdot \nabla \ . \tag{6.25}$$

Thus, we seek a stabilized method of the form: Find $u^h \in V^h \subset H_0^1(\Omega)$ such that $\forall v^h \in V^h$

$$(\kappa\nabla u^h, \nabla v^h) + (\mathbf{a} \cdot \nabla u^h, v^h) + \left(\mathcal{G}'(-\kappa\Delta u^h + \mathbf{a} \cdot \nabla u^h - f), \kappa\Delta v^h + \mathbf{a} \cdot \nabla v^h\right) = (f, v^h) \ , \tag{6.26}$$

where, as mentioned before, there is some loss of rigor in passing $\kappa\Delta v^h \in H^{-1} \not\subset L^2$ as an argument to the L^2 inner product, but we proceed formally nonetheless. (See [53, Section 3.2] for a more thorough treatment of terms like $\kappa\Delta v^h$ as distributions in the C^0 finite element setting.)

Supposing we want H_0^1 optimality, we introduce a localized approximation (cf. (6.22)),

$$u' \approx -\tau r^h \ , \tag{6.27}$$

where $\tau \geq 0$. The question then becomes: What is an appropriate choice of τ? In 1D, with $V^h = \text{CG}_1$, it is actually possible to derive a τ that makes the approximation exact. However, this derivation is quite tedious and does not generalize, applying only within a problem which could have been solved analytically to begin with. As such, we must resort to numerical analysis to find general-purpose bounds on τ, based on considerations of stability and consistency, much as we did earlier for PSPG.

Remarkably, this analysis was done well before the emergence of the VMS concept, in the context of a serendipitously-chosen stabilized formulation referred to as the **streamline-upwind Petrov–Galerkin (SUPG)** method. SUPG differs slightly from VMS, although the main features are the same. The SUPG formulation of advection–diffusion is: Find $u^h \in V^h$ such that for all $v^h \in V^h$,

$$(\kappa\nabla u^h, \nabla v^h) + (\mathbf{a} \cdot \nabla u^h, v^h) + \left(\tau(-\kappa\Delta u^h + \mathbf{a} \cdot \nabla u^h - f), \mathbf{a} \cdot \nabla v^h\right)_{\Omega'} = (f, v^h) \ . \tag{6.28}$$

We see that the term $\kappa\Delta v^h$ from the VMS formulation is absent, and we have replaced the formal L^2 inner product in the stabilization term with the $(\cdot,\cdot)_{\Omega'}$ product introduced in the PSPG discussion from Chapter 5, which is well-defined. However, the overall structure suggested by VMS reasoning is clearly present. The absence of $\kappa\Delta v^h$ can be justified when assuming $\mathbb{P}^h$ is the H_0^1 projector, by considering that $(u', \kappa\Delta v^h) = -\kappa(\nabla u', \nabla v^h) = 0$, i.e., u' is H_0^1-orthogonal to all $v^h \in V^h$.

Remark 45. The original heuristic reasoning behind SUPG was to introduce an anisotropic artificial diffusion along the flow direction of $\mathbf{a}$, augmenting the strong form's diffusion term like

$$-\kappa\Delta u = -\nabla \cdot (\kappa \mathbf{I} \nabla u) \quad \rightarrow \quad -\nabla \cdot ((\kappa \mathbf{I} + \tau(\mathbf{a} \otimes \mathbf{a}))\nabla u) \,. \tag{6.29}$$

One clearly sees, then, the connection with the artificial diffusion to which classical upwind finite difference methods are equivalent in 1D. However, unlike the weakly-consistent artificial diffusion formulation, SUPG remains *strongly*-consistent, and, for appropriate choices of τ, optimally-accurate.

Remark 46. The ad hoc replacement of our formal $(\cdot,\cdot)$ with $(\cdot,\cdot)_{\Omega'}$ leads to a well-defined numerical method, but what do we lose by ignoring the singular distributional contributions of the diffusive operator at element boundaries? Some research [55] has indicated that accuracy of stabilized methods can be improved by accounting for the distributional derivative information suggested by VMS but omitted by uncritically restricting formal integrals to element interiors. However, $(\cdot,\cdot)_{\Omega'}$ is sufficient for convergence analysis considered here and is widely-used in practical calculations.

We now summarize the stability and convergence analysis of SUPG originally published by Franca et al. in [56]. As in the analysis of PSPG from Chapter 5, our main tools will be a mesh-dependent norm, Young's inequality, and inverse estimates. We first isolate the bilinear part of the SUPG residual,

$$a_{\text{SUPG}}(u,v) := (\kappa\nabla u, \nabla v) + (\mathbf{a}\cdot\nabla u, v) + (\tau(-\kappa\Delta u + \mathbf{a}\cdot\nabla u)\,,\ \mathbf{a}\cdot\nabla v)_{\Omega'}\ . \tag{6.30}$$

We then introduce the norm $|||\cdot|||$ on V^h defined by

$$2|||u|||^2 = \kappa|u|^2_{H^1} + \left\| \sqrt{\tau}\,\mathbf{a}\cdot\nabla u \right\|^2_{L^2}\ . \tag{6.31}$$

For what choices of τ is a_{SUPG} coercive in this norm? For simplicity, let us assume that V^h is defined by a quasi-uniform mesh of element diameter h. (The arguments below can be generalized easily to accommodate a different h on each element, by splitting all integrals into sums over elements.) We then follow a similar path as the analysis of PSPG, assuming $v \in V^h$ (and thus subject to inverse estimates):

$$\begin{aligned} a_{\text{SUPG}}(v,v) &= 2|||v|||^2 + \tau(-\kappa\Delta v, \mathbf{a}\cdot\nabla v)_{\Omega'} && (6.32)\\ &\geq 2|||v|||^2 - \left(\frac{\kappa}{2}\left\|\sqrt{\tau}\Delta v\right\|^2_{\Omega'} + \frac{1}{2}\left\|\sqrt{\tau}\mathbf{a}\cdot\nabla v\right\|^2_{L^2}\right)\ . && (6.33)\end{aligned}$$

Using the inverse estimate (cf. (2.34))

$$\|\Delta v\|^2_{L^2(\Omega_e)} \leq C_{\text{inv}} h^{-2} \|\nabla v\|^2_{L^2(\Omega_e)} \quad \forall v \in V^h\,, \tag{6.34}$$

then

$$a_{\text{SUPG}}(v,v) \geq \kappa \left| v\sqrt{1 - \frac{1}{2}\tau\kappa C_{\text{inv}} h^{-2}} \right|^2_{H^1} + \frac{1}{2}\left\|\sqrt{\tau}\mathbf{a}\cdot\nabla v\right\|^2_{L^2}\ . \tag{6.35}$$

Thus, for

$$\tau \leq \frac{h^2}{\kappa C_{\text{inv}}}\,, \tag{6.36}$$

the form a_{SUPG} is coercive with unit constant in the norm $|||\cdot|||$. However, unlike the case of PSPG, this condition alone will not be sufficient to ensure optimal convergence with a nice constant. We nonetheless proceed as in the analysis of PSPG, first defining the error and its split into interpolation and method parts,

$$e := u - u^h = e^h + \eta \, , \tag{6.37}$$

where $e^h \in V^h$.

Remark 47. In the VMS picture, e has the interpretation of u', and we would like for it to be bounded in the H^1 seminorm in a way that does not depend on the coefficients of the problem (which our desired projector is not informed by).

Next, as in the PSPG analysis, we use this coercivity to bound $\left|\left|\left|e^h\right|\right|\right|^2$, then proceed to expand a_{SUPG} and apply Young's inequality and inverse estimates, but without yet committing to a particular choice of τ:

$$\begin{aligned}
\left|\left|\left|e^h\right|\right|\right|^2 &\leq a_{\text{SUPG}}(e^h, e^h) && (6.38)\\
&= a_{\text{SUPG}}(-\eta, e^h) && (6.39)\\
&\leq -\left(\kappa(\nabla\eta, \nabla e^h) - (\eta, \mathbf{a}\cdot\nabla e^h) + \left(\tau(-\kappa\Delta\eta + \mathbf{a}\cdot\nabla\eta), \mathbf{a}\cdot\nabla e^h\right)_{\Omega'}\right) && (6.40)\\
&\leq \frac{1}{2}\left(\frac{\kappa}{\epsilon_1}\|\nabla\eta\|^2 + \kappa\epsilon_1\left\|\nabla e^h\right\|^2 + \frac{1}{\epsilon_2}\left\|\tau^{-1/2}\eta\right\|^2 + \epsilon_2\left\|\sqrt{\tau}\mathbf{a}\cdot\nabla e^h\right\|^2\right.\\
&\qquad \left. + \frac{1}{\epsilon_3}\left\|\kappa\sqrt{\tau}\Delta\eta\right\|^2_{\Omega'} + \epsilon_3\left\|\sqrt{\tau}\mathbf{a}\cdot\nabla e^h\right\|^2 + \frac{1}{\epsilon_4}\left\|\sqrt{\tau}\mathbf{a}\cdot\nabla\eta\right\|^2 + \epsilon_4\left\|\sqrt{\tau}\mathbf{a}\cdot\nabla e^h\right\|^2\right) \, . && (6.41)
\end{aligned}$$

To effectively "hide" the $\|\nabla e^h\|^2$ terms, we can take $\epsilon_1 = 1/2$ and

$$\epsilon_2 + \epsilon_3 + \epsilon_4 = 1/2 \, . \tag{6.42}$$

Supposing that we choose

$$\epsilon_2 = \epsilon_3 = \epsilon_4 = 1/6 \, , \tag{6.43}$$

we get

$$\frac{1}{2}\left|\left|\left|e^h\right|\right|\right|^2 \leq \kappa\|\nabla\eta\|^2 + 3\left\|\tau^{-1/2}\eta\right\|^2 + 3\left\|\kappa\sqrt{\tau}\Delta\eta\right\|^2_{\Omega'} + 3\left\|\sqrt{\tau}\mathbf{a}\cdot\nabla\eta\right\|^2 \, . \tag{6.44}$$

Based on this result, we add the following additional condition on the stabilization parameter:

$$\tau \leq \frac{h}{2|\mathbf{a}|} \, . \tag{6.45}$$

To satisfy both restrictions, we take

$$\tau = \min\left\{\frac{h^2}{\kappa C_{\text{inv}}}, \frac{h}{2|\mathbf{a}|}\right\} \, . \tag{6.46}$$

We assume here that C_{inv} is bounded below by some moderate nonzero constant.[1]

[1]For CG$_1$ simplex elements, this is not a sharp choice of inverse estimate constant, since (6.34) is satisfied for $C_{\text{inv}} = 0$. However, the consistency analysis that follows assumes $C_{\text{inv}} > 0$. See the analysis of [56] for a more precise construction.

Then, leveraging both inverse and interpolation estimates in a CG$_k$ FE space, we get, for sufficiently-smooth u, that η can be chosen to show that $\left|\left|\left|e^h\right|\right|\right|$ converges to zero as $h \to 0$. Let us look more closely at this result, in the 1D constant-coefficient case, with $\mathbf{a} = a\mathbf{e}_1$, for $a \in \mathbb{R}$. Then we have

$$\left(\kappa + \tau a^2\right) \left\|e^h_{,x}\right\|^2 \leq C\left(\kappa + \frac{h^2}{\tau} + \frac{\kappa^2\tau}{h} + \tau a^2\right) h^{2k} \,, \tag{6.47}$$

so, in the diffusive limit of $\tau = h^2/(C_{\text{inv}}\kappa)$ and $|a| \leq C\kappa/h$, we get

$$\kappa \left\|e^h_{,x}\right\|^2 \leq C\kappa h^{2k} \quad \Rightarrow \quad \left|e^h\right|_{H^1} \leq Ch^k \,, \tag{6.48}$$

and, in the advective limit of $\tau = h/(2|a|)$ and $\kappa \leq C|a|h$, we get

$$h|a| \left\|e^h_{,x}\right\|^2 \leq C(h|a|)h^{2k} \quad \Rightarrow \quad \left|e^h\right|_{H^1} \leq Ch^k \,, \tag{6.49}$$

where the constants denoted by "C" in the above may depend on (derivatives of) the exact solution u, but are **independent of κ and** a, i.e., we recover H^1_0 optimality up to a constant that is unrelated to our choice of coefficients.

Remark 48. It is easy to see that, with V^h = CG$_1$ in 1D, the advective limit with no source term recovers the weakly-consistent artificial diffusion scheme that was shown in an earlier homework exercise to be equivalent to an upwind finite difference scheme. Unlike upwind finite differences or artificial diffusion, though, SUPG remains optimally-accurate for smooth solutions. (When it is said in some references that upwind differences or artificial diffusion are "first order", this refers to convergence in the L^2 norm, where first order accuracy is sub-optimal.)

Remark 49. An error analysis of SUPG that is more in line with the standard use of the inf-sup theorem can be found in [57].

6.4 FEniCS demonstration of SUPG

To illustrate this numerically, we consider the classic 1D boundary-layer problem of advection–diffusion using FEniCS: Find u such that

$$\begin{cases} -\kappa u''(x) + au'(x) = 0 \quad \forall x \in (0,1) \\ u(0) = 0 \\ u(1) = 1 \end{cases} \tag{6.50}$$

where $\kappa > 0$ and $a > 0$. The exact solution is

$$u(x) = \frac{e^{ax/\kappa} - 1}{e^{a/\kappa} - 1} \,. \tag{6.51}$$

One sees, immediately, that there is a sharp **boundary layer** near $x = 1$ if $a/\kappa \gg 1$. This is the fine-scale solution behavior that is not being resolved. The FEniCS implementation is as follows:

```
from dolfin import *
from ufl import Min

# Setup:
N = 8
k = 1
# Setting to False on coarse meshes shows
# the poor performance of Bubnov--Galerkin.
use_SUPG = True
# Setting kappa too low eventually produces
# numerical stability issues in the exact
# solution.
kappa = Constant(5e-3)
a = Constant((1,))

# Formulation:
mesh = UnitIntervalMesh(N)
V = FunctionSpace(mesh,"CG",k)
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(0)
res_Gal = (kappa*inner(grad(u),grad(v))
           + dot(a,grad(u))*v - f*v)*dx
res_strong = -div(kappa*grad(u)) + dot(a,grad(u)) - f
h = CellDiameter(mesh)
Cinv = Constant(6.0*k*k)
# (See the Brooks & Hughes reference for a nodally-
# exact tau, tuned for this 1D problem; we use
# the more generic result from numerical analysis
# in this script.)
tau = Min(h*h/(Cinv*kappa),h/(2*sqrt(dot(a,a))))
res_SUPG = tau*res_strong*dot(a,grad(v))*dx
res = res_Gal
if(use_SUPG):
    res += res_SUPG

# Solve:
uh = Function(V)
solve(lhs(res)==rhs(res),uh,
      [DirichletBC(V,Constant(0),"near(x[0],0)"),
       DirichletBC(V,Constant(1),"near(x[0],1)")])

# Check error:
x = SpatialCoordinate(mesh)[0]
# (Use Expression to be able to interpolate in plotting)
```

```
ex_deg = k+3
u_ex = Expression("(exp(a*x[0]/k)-1)/(exp(a/k)-1)",
                  degree=ex_deg,domain=mesh,
                  a=float(a[0]),k=float(kappa))
e = interpolate(uh,FunctionSpace(mesh,"CG",ex_deg))-u_ex
print("H1 seminorm error = "
      +str(sqrt(assemble(dot(grad(e),grad(e))*dx))))
print("L2 error = "
      +str(sqrt(assemble(e*e*dx))))

# Plot:
from matplotlib import pyplot as plt
plot(uh)
# (Put exact solution on refined mesh)
mesh_interp = UnitIntervalMesh(1000)
V_interp = FunctionSpace(mesh_interp,"CG",1)
u_ex_interp = interpolate(u_ex,V_interp)
plot(u_ex_interp)
plt.autoscale()
plt.show()
```

By running and modifying this script, it is easy to see that SUPG stabilization leads to a much more accurate approximation than Galerkin's method (invoked with `use_SUPG=False`) when $ah/\kappa \gg 1$, i.e., the boundary layer fine-scale feature is under-resolved.

Remark 50. Even with SUPG, optimal approximation will not yield optimal convergence rates until h is sufficiently small to resolve fine-scale features, i.e., $ah/\kappa \lesssim 1$. However, stabilization prevents errors from blowing up too quickly as $ah/\kappa \to \infty$. SUPG also allows for accurate approximation in regions of the domain where the solution is slowly-varying. The deleterious effects of unresolved features on Galerkin's method are seen throughout the domain; this is referred to as a **pollution effect** in the numerical analysis literature.

6.5 VMS interpretation of PSPG/LSIC

It is clear from Section 6.3 that there are close formal ties between SUPG and PSPG, and PSPG is sometimes considered "SUPG for incompressibility". This leads one to ask: Can PSPG also be motivated by VMS reasoning? The answer is yes. Taking the forms a and b from Chapter 5, it is easy to see how the VMS formalism leads to the coarse-scale problem

$$a\left(\mathbf{u}^h + \mathbf{u}', \mathbf{v}^h\right) + b\left(\mathbf{u}^h + \mathbf{u}', q^h\right) - b\left(\mathbf{v}^h, p^h + p'\right) = L\left(\mathbf{v}^h\right) . \tag{6.52}$$

To obtain PSPG from this, assume $a(\mathbf{u}', \mathbf{v}^h) = 0$ is the optimality condition for $\mathbf{u}^h$, formally integrate by parts in $b(\mathbf{u}', q^h)$, then make the localized approximations

$$\mathbf{u}' \approx -\tau_{\mathrm{M}}\left(-\nabla\cdot(2\mu\varepsilon(\mathbf{u}^h)) + \nabla p^h - \mathbf{f}\right) \tag{6.53}$$

$$q' \approx -\tau_{\mathrm{C}}\nabla\cdot\mathbf{u}^h \tag{6.54}$$

for the action of the fine-scale Green's operator on the residuals of the momentum and continuity equations.

6.6 Navier–Stokes and turbulence modeling

We now consider an extrapolation of the VMS theory to the nonlinear Navier–Stokes problem. The theory was developed by Bazilevs et al. [58], based on formal perturbation arguments and earlier ideas of Hughes et al. [59]. The resulting formulation is: Find $(\mathbf{u}^h, p^h) \in V^h \times Q^h$ such that $\forall(\mathbf{v}^h, q^h) \in V^h \times Q^h$,

$$F_{\text{Gal}}((\mathbf{u}^h, p^h), (\mathbf{v}^h, q^h)) + F_{\text{SUPG}}((\mathbf{u}^h, p^h), (\mathbf{v}^h, q^h)) + F_{\text{VMS}}((\mathbf{u}^h, p^h), (\mathbf{v}^h, q^h)) = 0 , \tag{6.55}$$

where, reusing definitions of a, b, and $\boldsymbol{\sigma}$ from Stokes flow,

$$F_{\text{Gal}}((\mathbf{u}, p), (\mathbf{v}, q)) = (\rho(\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}) - \mathbf{f}, \mathbf{v}) + a(\mathbf{u}, \mathbf{v}) + b(\mathbf{u}, q) - b(\mathbf{v}, p) \tag{6.56}$$

is from the Galerkin weak form of the problem,

$$F_{\text{SUPG}}((\mathbf{u}, p), (\mathbf{v}, q)) = \left(\mathbf{u}', -(\rho\mathbf{u} \cdot \nabla \mathbf{v} + \nabla q)\right)_{\Omega'} \tag{6.57}$$

with

$$\mathbf{u}' = -\tau_{\text{M}} \left(\rho(\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}) - \nabla \cdot \boldsymbol{\sigma}(\mathbf{u}, p) - \mathbf{f}\right) \tag{6.58}$$

are the terms following from the Navier–Stokes generalization of SUPG (which includes PSPG), and

$$F_{\text{VMS}}((\mathbf{u}, p), (\mathbf{v}, q)) = -b(\mathbf{v}, p') + (\rho\mathbf{u}' \cdot \nabla \mathbf{u}, \mathbf{v})_{\Omega'} - (\rho\mathbf{u}' \otimes \mathbf{u}', \nabla \mathbf{v})_{\Omega'} \tag{6.59}$$

with

$$p' = -\tau_{\text{C}} \nabla \cdot \mathbf{u} \tag{6.60}$$

are further terms following from VMS logic applied to the fine-scale pressure and nonlinear convection term. A choice of τ_{M} motivated by numerical analysis considerations would have the overall behavior

$$\tau_{\text{M}} \sim \min\left\{\frac{\Delta t}{2\rho}, \frac{h}{2|\rho\mathbf{u}|}, \frac{h^2}{C_{\text{inv}}\mu}\right\} \tag{6.61}$$

with τ_{C} following as in PSPG. One way to understand the term involving time step Δt is to take a space–time viewpoint, and consider the partial time derivative to be part of a space–time advection operator. In practice, though, stabilization constants for this formulation are usually "smoothed" versions of this, and employ some more sophisticated notion of "h". Formulas that have been applied successfully in many applications are

$$\tau_{\text{M}} = \left(\left(\frac{2\rho}{\Delta t}\right)^2 + \rho\mathbf{u} \cdot (\mathbf{G}\rho\mathbf{u}) + (C_{\text{inv}}\mu)^2\, \mathbf{G} : \mathbf{G}\right)^{-1/2} , \tag{6.62}$$

$$\tau_{\text{C}} = (\tau_{\text{M}} \operatorname{tr} \mathbf{G})^{-1} , \tag{6.63}$$

where the tensor $\mathbf{G}$ is given by

$$G_{ij} = \frac{\partial \xi_i}{\partial x_k} \frac{\partial \xi_j}{\partial x_k} \quad \left(\sim h^{-2} \right) , \tag{6.64}$$

in which $\boldsymbol{\xi}$ are coordinates of physical points pulled back to the reference element, under the assumption that this reference element is a *bi*-unit hyper-square, i.e., $(-1, 1)^d$. (For simplicial elements parameterized with a unit edge on each parameteric coordinate axis, as assumed in FEniCS's FIAT component, one would scale $\mathbf{G}$ by $1/4$.)

Remarkably, [58] and many subsequent papers—on both fundamentals of turbulence [60, 61, 62] and industrial applications [63, 64]—have found that variants of (6.55) are effective not just as stabilization, but also as an implicit large-eddy simulation (LES) filter, i.e., that it is a viable turbulence simulation technology emerging entirely from considerations of numerical analysis, without any additional "turbulence modeling" beyond the Navier–Stokes equations themselves. FEniCS-based implementations of this formulation is given in the small library VarMINT [65], and also in the more comprehensive solver IlliniFlow [66, 67].

Remark 51. The formulation (6.55) is sometimes also augmented with a discontinuity capturing (DC) term, of the form

$$F_{\mathrm{DC}}((\mathbf{u}, p), (\mathbf{v}, q)) = ((\mathbf{u}' \cdot \nabla\mathbf{u}), \overline{\tau}(\mathbf{u}' \cdot \nabla\mathbf{v}))_{\Omega'} \qquad (6.65)$$

where

$$\overline{\tau} = \rho\left(\mathbf{u}' \cdot \mathbf{G}\mathbf{u}' + \epsilon^2\right)^{-1/2} , \qquad (6.66)$$

with ϵ being some small regularization near machine zero, to avoid divide-by-zero errors (which would otherwise occur when $\mathbf{u}' = \mathbf{0}$, as is common in, e.g., all-zero initial conditions). This term is an ad hoc device to add dissipation near sharp layers (while remaining strongly-consistent), but some recent work has attempted to relate DC operators to the VMS picture [68].

Remark 52. For high Reynolds number wall-bounded flows, the strongly-enforced Dirichlet boundary conditions implied by taking $V = (H^1_0(\Omega))^d$ require highly-refined boundary layer meshes to be accurate, even with the VMS formulation of this section. On unstretched meshes, good results are only obtained by using weakly-enforced Dirichlet boundary conditions, as noted in some of the references cited above. On coarse boundary layer discretizations, weakly-enforced Dirichlet conditions behave analogously to the "wall models" used in classical turbulence simulation, while remaining strongly-consistent with no-slip, and converging optimally to that in the limit of $h \to 0$. We will cover weak enforcement of Dirichlet conditions in Section 7.4. Weak Dirichlet boundary conditions were recently derived from VMS considerations in [69].

Remark 53. Researchers have also reached the conclusion that residual-based stabilized formulations are effective for turbulent flow through research programs independent of VMS. For instance, Hoffman and Johnson [60] consider a broadly-similar stabilized formulation dubbed General Galerkin (G2) which, in conjunction with mesh adaptivity, they promote as a solution to turbulence. Their framework has been applied with great success (using a modified fork of FEniCS [70]) to challenging applied problems, including stall prediction for a full-scale airplane [71].

Remark 54. An analysis similar to that performed for SUPG stabilization of advection–diffusion in Section 6.3 can be carried out for SUPG of a linearized version of Navier–Stokes called Oseen flow [72].

Remark 55. Work on the extension of SUPG/VMS to compressible Navier–Stokes flow is reviewed in [73]. Augmenting VMS with a DC term is essential in that setting, where shocks can develop in the solution.

Chapter 7

Coupling between domains: The fluid–structure interaction kinematic constraint

7.1 Introduction

Previous chapters have covered the main numerical issues in discretizing the fluid and structure subproblems in FSI, highlighting the fact that the fluid subproblem involves coupling between fields and length scales. We are now poised to discuss the discretization of FSI itself, which involves a qualitatively-different form of coupling, namely, coupling between PDE systems posed on disjoint subdomains. As in the case of incompressibility, this will include a constraint on the kinematics of the coupled problem, which will be formulated in Section 7.2 using a Lagrange multiplier field in the variational setting, this time restricted to the interface between the fluid and structure subdomains.

As with incompressibility, the discretization of a saddle point problem brings with it the difficulty of needing to satisfy a discrete inf-sup condition. If the fluid–structure interface were built directly into a mesh of the full domain, one might have some hope of designing a function space for the Lagrange multiplier which would maintain uniform inf-sup stability in Bubnov–Galerkin discretizations, but, in that case, it would be more typical to simply discretize the problem in a function space that conforms to the constraint, as detailed in Section 7.3 (and analogous to the divergence-conforming discretizations of incompressibility mentioned in Sections 5.3.2 and 5.3.3). However, constructing a conforming mesh—especially one that is robust to large deformations of the domain—may become prohibitively difficult. Section 7.4 considers a residual-based stabilized method analogous to PSPG (Section 5.4), which is amenable to non-matching fluid and structure meshes. We find, in that case, that the stabilized multiplier field can be formally eliminated from the discrete problem, in a way that is analogous to the classical boundary condition enforcement technique known in the literature as "Nitsche's method".

7.2 The augmented Lagrangian formulation of FSI

In this section, we summarize the FSI formulation proposed in [74], which generates the methods detailed in Sections 7.3 and 7.4.3 as special cases. For simplicity, we proceed with a treatment that is most directly applicable to incompressible Navier–Stokes flow for the fluid subproblem and compressible hyperelasticity for the solid subproblem, but the ideas related to the coupling of these problems generalize in a straightforward way to other models.

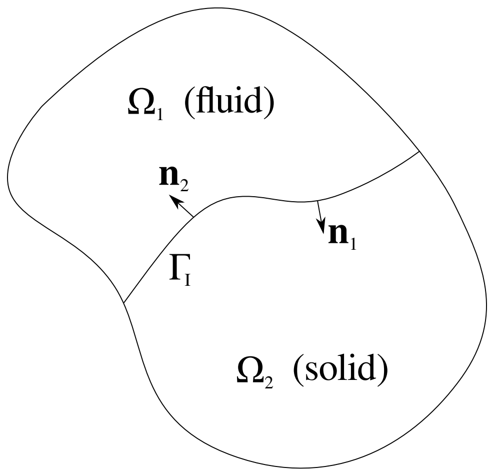

Figure 7.1: Division of Ω_x into fluid and solid subdomains Ω_1 and Ω_2, meeting at the fluid–solid interface, Γ_I.

We begin by considering an instance of the generalized continuum mechanics problem of Chapter 4, where the spatial domain Ω_x is divided into two subdomains, Ω_1 and Ω_2, on which the fluid and structure subproblems—which we will continue to index by 1 and 2, respectively—are posed. This decomposition is illustrated in Figure 7.1. In the absence of fluid–structure coupling, these would be: Find fluid velocity $\mathbf{v}_1 \in V_1$ and pressure $p \in Q$ such that, for all test functions $\mathbf{w}_1 \in V_1$ and $q \in Q$

$$N_1(\{\mathbf{v}_1, p\}, \{\mathbf{w}_1, q\}) = 0 \tag{7.1}$$

and: Find structure displacement $\mathbf{u}_2 \in V_2$ such that, for all test functions $\mathbf{w}_2 \in V_2$

$$N_2(\mathbf{u}_2, \mathbf{w}_2) = 0 \ , \tag{7.2}$$

where V_i is the function space for the velocity/displacement solution of subproblem i, Q is the function space for the fluid pressure, and N_i is the variational form of the nonlinear residual for subproblem i. The FSI problem can then be written: Find $\{\mathbf{v}_1, p\} \in V_1 \times Q$, $\mathbf{u}_2 \in V_2$, and $\boldsymbol{\lambda} \in V_\lambda$

such that, for all $\{\mathbf{w}_1, q\} \in V_1 \times Q$, $\mathbf{w}_2 \in V_2$, and $\delta\boldsymbol{\lambda} \in V_\lambda$,

$$
\begin{aligned}
&N_1(\{\mathbf{v}_1, p\}, \{\mathbf{w}_1, q\}) + N_2(\mathbf{u}_2, \mathbf{w}_2) \\
&\quad + \int_{\Gamma_\mathrm{I}} \boldsymbol{\lambda} \cdot (\mathbf{w}_1 - \mathbf{w}_2)\, d\Gamma_\mathrm{I} \\
&\quad + \int_{\Gamma_\mathrm{I}} \delta\boldsymbol{\lambda} \cdot (\mathbf{v}_1 - \mathbf{v}_2)\, d\Gamma_\mathrm{I} \\
&\quad + \int_{\Gamma_\mathrm{I}} \beta(\mathbf{v}_1 - \mathbf{v}_2) \cdot (\mathbf{w}_1 - \mathbf{w}_2)\, d\Gamma_\mathrm{I} = 0\,,
\end{aligned}
\tag{7.3}
$$

where $\mathbf{v}_2 = \partial_s \mathbf{u}_2$ is the solid velocity field, Γ_I is the shared boundary $\overline{\Omega}_1 \cap \overline{\Omega}_2$ of the fluid and solid subdomains and the new unknown $\boldsymbol{\lambda}$ is a Lagrange multiplier field on Γ_I, to enforce the constraint that $\mathbf{v}_1 = \mathbf{v}_2$. The coefficient $\beta > 0$ is a **penalty parameter**; the corresponding term is zero if the constraint is enforced exactly, but this term plays a role in some numerical schemes, where the constraint may not be satisfied pointwise.

Remark 56. The variational problem (7.3) clearly fits the template of a saddle-point problem (cf. (5.3)), and the inf-sup stability concerns discussed in the context of incompressibility (Chapter 5) apply here as well.

FSI coupling is typically stated as *two* conditions, though: not just the kinematic constraint enforced above, but also the traction compatibility condition

$$
\boldsymbol{\sigma}_1 \cdot \mathbf{n}_1 = -\boldsymbol{\sigma}_2 \cdot \mathbf{n}_2\,, \tag{7.4}
$$

where $\boldsymbol{\sigma}_i$ is the Cauchy stress in subproblem i and $\mathbf{n}_1 = -\mathbf{n}_2$ are the outward-facing normals to the subdomains. The condition (7.4) is actually built into the augmented Lagrangian formulation. If we assume that both N_1 and N_2 have the form

$$
N_i = \cdots + \int_{\Omega_i} \boldsymbol{\sigma}_i : \nabla_x \mathbf{w}_i \, d\Omega_i + \cdots\,, \tag{7.5}
$$

as is typical of weak formulations used in finite element methods, then integrating the emphasized term by parts and comparing with (7.3) shows that

$$
-\boldsymbol{\sigma}_1 \cdot \mathbf{n}_1 = \boldsymbol{\sigma}_2 \cdot \mathbf{n}_2 = \boldsymbol{\lambda} + \beta(\mathbf{v}_1 - \mathbf{v}_2)\,, \tag{7.6}
$$

i.e., the Lagrange multiplier and penalty forcing act as tractions on the solid subproblem and their negations act as a tractions on the fluid subproblem, thereby satisfying (7.4). If this line of reasoning is unclear, recall Exercise 10 from Chapter 1 for an analogous argument in a simpler problem setting.

7.3 Conforming FSI

One way to circumvent the difficulties associated with solving a saddle-point problem is to restrict the solution space to a subset satisfying the constraint. In the case of incompressibility, it is difficult

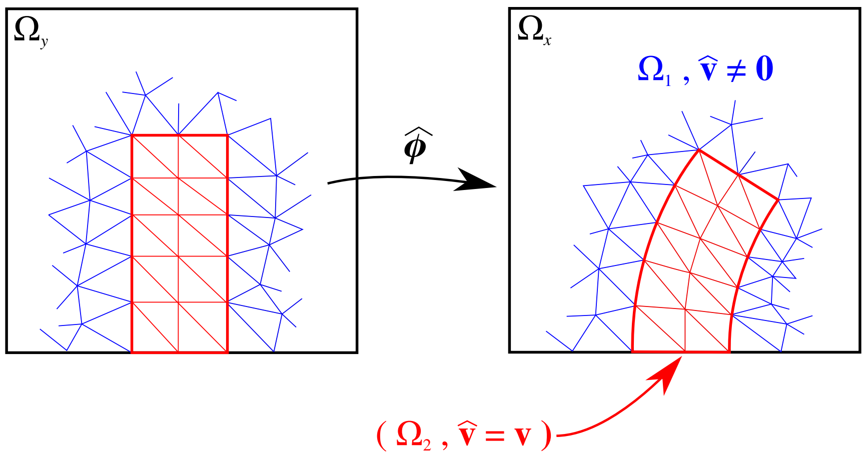
Ω_y
$\widehat{\phi}$
Ω_x
Ω_1 , $\hat{\mathbf{v}} \neq \mathbf{0}$
(Ω_2 , $\hat{\mathbf{v}} = \mathbf{v}$)

Figure 7.2: Illustration of conforming FSI. The structure, discretized by the red mesh, uses the Lagrangian description, while the fluid, discretized by the blue mesh, uses the general ALE description, where $\hat{\mathbf{v}}$ is determined by solving an auxiliary mesh-motion subproblem.

to explicitly construct this space. However, in the case of FSI, it reduces to constructing a finite element mesh of Ω_x that conforms to Γ_I, as illustrated in Figure 7.2. In some cases, where the structure has a simple geometry and undergoes mild motion, this is straightforward to do. The current section covers such discretizations in detail, including implementation with FEniCS.

If a conforming mesh of Ω_x is available, then finite-dimensional subsets of V_1 and V_2 can be constructed such that $\mathbf{v}_1 = \mathbf{v}_2$ and $\mathbf{w}_1 = \mathbf{w}_2$ on Γ_I. In that case, all boundary terms of (7.3) are zero. The challenge, however, is that Ω_x and Γ_I are time-dependent. To address this challenge, we work within Chapter 4's framework for conservation laws on deforming domains. The most expedient approach to obtain conforming fluid and structure function spaces is to construct them once on a static reference domain, Ω_y, with $\hat{\mathbf{v}} = \mathbf{v}$ on the subset corresponding to the solid subproblem, so that the velocity of Γ_I will be $\hat{\mathbf{v}}$, and the mesh of Ω_y will track the fluid–structure interface as it deforms. In most problems, $\hat{\phi}$ in Ω_1 will then continuously interpolate between the material motion ϕ in Ω_2 and some external boundaries of Ω_1 which do not follow the material. This leads to a nonzero $\hat{\mathbf{v}}$ throughout Ω_1, which is not uniquely determined by the physics of the FSI problem; practical methods for constructing such a $\hat{\phi}$ remain an active area of research in computational FSI, although the most common is to solve a fictitious elasticity problem on Ω_1, as will be done in the example implementation of this section.

Because a complete nontrivial code example is dominated by problem-specific setup of the domain geometry and boundary conditions, we summarize the parts of the implementation formulating the problem here, with a full program in Appendix C. As such, we begin directly with the definition of function spaces:

```
cell = mesh.ufl_cell()
Ve = VectorElement("CG", cell, 1)
Qe = FiniteElement("CG", cell, 1)
VQe = MixedElement((Ve,Qe))
W = FunctionSpace(mesh,VQe)
V = FunctionSpace(mesh,Ve)
```

where `mesh` is a DOLFIN `Mesh` object, partitioning the static reference domain Ω_y into elements, to define finite element function spaces. The space `W` is a mixed space for the combined fluid–structure velocity field and the fluid pressure, while `V` is a function space for the displacement field

$$\hat{\mathbf{u}}(\mathbf{y}) := \hat{\boldsymbol{\phi}}(\mathbf{y}) - \mathbf{y} \tag{7.7}$$

associated with the mapping $\hat{\boldsymbol{\phi}}$ from Ω_y to Ω_x. In the solid region, we shall assume a Lagrangian description where $\boldsymbol{\phi} = \hat{\boldsymbol{\phi}}$, but, in the fluid region, an ALE description is used. Next, we define the `Function` objects for unknowns at the current and previous time levels:

```
# Mesh motion functions:
uhat = Function(V)
uhat_old = Function(V)
du = TestFunction(V)

# Fluid--structure functions:
(dv, dp) = TestFunctions(W)
w = Function(W)
v,p = split(w)
w_old = Function(W)
v_old, p_old = split(w_old)
```

Notably absent is the structure's displacement, which is typically used to formulate the solid subproblem. This will be defined symbolically, through the choice of a time integration scheme. In particular, we select the backward Euler method, for simplicity, although it is straightforward to extend this approach to more accurate implicit integrators (cf. Exercise 3b in Chapter 3):

```
vhat = (uhat-uhat_old)/Dt
dv_dr = (v - v_old)/Dt
dv_ds = dv_dr
u = uhat_old + Dt*v
```

Note that, because the finite element mesh and function spaces are defined on Ω_y, the finite differences in time here correspond to the partial derivative ∂_r or, when restricted to the solid subdomain, ∂_s. The solid displacement `u` at the current time level is defined in the final line of the above listing.

Next, we define the necessary spatial partial derivatives:

```
dX = dx(SOLID_FLAG)
dy = dx
grad_y = grad
grad_X = grad
y = SpatialCoordinate(mesh)
x = y + uhat
det_dxdy = det(grad_y(x))
def grad_x(f):
    return dot(grad_y(f),inv(grad_y(x)))
def div_x(f): # (For vector-valued f)
    return tr(grad_x(f))
```

```
def div_x_tens(f): # (For (rank-2) tensor-valued f)
    i,j = indices(2)
    return as_tensor(grad_x(f)[i,j,j],(i,))
```

where SOLID_FLAG marks a the solid subdomain of Ω_y. Note that there is a potentially-confusing notational conflict between UFL and Chapter 4, where the symbol “x” in UFL is used generically as a coordinate in whatever domain the mesh occupies, which, in this example is Ω_y. Thus, we have aliased UFL’s dx integration measure accordingly. The determinant det_dxdy corresponds to $\hat{J}^{-1}$ in the notation of Chapter 4; for technical reasons, it is not straightforward to define a UFL measure for Ω_x, so we shall simply use det_dxdy*dy in variational forms.

Recall that the mapping $\hat{\phi}$ in the fluid subdomain is *arbitrary*, and not uniquely determined by the physics of the problem. In this example, we define an auxiliary elasticity-like PDE which it will satisfy:

```
F_m = grad_y(uhat) + I
E_m = 0.5*(F_m.T*F_m - I)
m_jac_stiff_pow = Constant(3)
# Artificially stiffen the mesh where it is getting crushed:
K_m = Constant(1)/pow(det(F_m),m_jac_stiff_pow)
mu_m = Constant(1)/pow(det(F_m),m_jac_stiff_pow)
S_m = K_m*tr(E_m)*I + 2.0*mu_m*(E_m - tr(E_m)*I/3.0)
res_m = (inner(F_m*S_m,grad_y(du)))*dy
```

where I is the identity tensor and res_m is the nonlinear residual form of the auxiliary problem problem. While formally posed over all of Ω_y, as indicated by the integration measure dy, the solution uhat will be constrained by a Dirichlet boundary condition in the solid subdomain, details of which can be seen in Appendix C. The deformation gradient F_m corresponds to $\hat{\mathbf{F}}$. The stiffening of the artificial mesh material where $\hat{J}$ is small is called Jacobian-based mesh stiffening, and comes from the pioneering work of Tezdyuar and collaborators [75, 76, 77, 78].

The nonlinear residual of the solid subproblem, assuming a St. Venant–Kirchhoff constitutive model, is given by

```
mu_s = Constant(1e4)
K = Constant(1e4)
rho_s0 = Constant(1)
F = grad_X(u) + I
E = 0.5*(F.T*F - I)
S = K*tr(E)*I + 2.0*mu_s*(E - tr(E)*I/3.0)
f_s = Constant(d*(0,))
res_s = rho_s0*inner(dv_ds - f_s,dv)*dX \
        + inner(F*S,grad_X(dv))*dX
```

and the Galerkin part of the fluid subproblem residual is

```
rho_f = Constant(1)
mu_f = Constant(1e-2)
sigma_f = 2.0*mu_f*sym(grad_x(v)) - p*I
v_adv = v - vhat
```

```
DvDt = dv_dr + dot(grad_x(v),v_adv)
resGal_f = (rho_f*dot(DvDt,dv) + inner(sigma_f,grad_x(dv))
            + dp*div_x(v))*det_dxdy*dy(FLUID_FLAG)
```

where FLUID_FLAG indicates the fluid subdomain of Ω_y. The pressure field p will be constrained by a Dirichlet boundary condition on all interior degrees of freedom of the solid subdomain. Recall that, in the solid subproblem, the unknown solution is in fact the same velocity v used to formulate the fluid subproblem residual, which is used in the definition of the symbol u, along with uhat_old (according to the implicit time integration scheme). Thus, the residuals res_s and resGal_f can be added together for a single monolithic problem, to be solved for the current time level's velocity and pressure unknowns, v and p, which are components of a Function, w, in the mixed space W. The code in Appendix C also defines stabilization residuals for SUPG/PSPG and LSIC terms, to be added to resGal_f for a robust spatial discretization, leading to the combined residual

```
res_f = resGal_f + resSUPG_f + resLSIC_f + resOutflow_f
res_fs = res_f + res_s
```

where resOutflow_f is a further term adding a Neumann boundary condition for the specific problem instance given in the appendix.

In principle, the mesh displacement uhat could be a further component of the mixed space W, and the full fluid–structure–mesh problem could be solved in a single monolithic system. However, this is inefficient, as it is usually stable to alternate between the smaller mesh motion and continuum mechanics problems.[1] Thus the solution strategy implemented in Appendix C is then to iterate the following steps within each implicit time step until res_fs assembles to a vector meeting some convergence criterion:

1. Solve the nonlinear variational problem res_fs == 0 for v and p, holding the mesh deformation uhat fixed.

2. Solve the nonlinear variational problem res_m == 0 for uhat, with a boundary condition that uhat restricted to the solid subdomain is fixed to u, which is defined in terms of a fixed v, solved for in Step 1.

The solution algorithm given in Appendix C is mainly designed for code clarity, and is unlikely to provide optimal performance in a given practical problem. Many variations on such solution strategies exist in the literature, such as performing just a single Newton iteration in Step 1 and solving a linear problem for the mesh's deformation within the current time step in Step 2, and, as discussed further in Chapter 9, some physical situations may warrant splitting the fluid and structure solves into separate steps. In the computational fluid dynamics community, it is even popular to split the incompressible flow subproblem into multiple steps, to further reduce the sizes of linear systems.

7.4 Weak boundary conditions and Nitsche's method

As mentioned in Section 7.3, it may be difficult to generate a conforming mesh like the one illustrated in Figure 7.2. Alternatively, the spatial resolutions appropriate to the subproblems may

[1]Further, the boundary condition on the mesh motion subproblem would not be straightforward to enforce through FEniCS's abstractions.

be very different. In such cases, one may wish to instead use non-matching meshes, like those depicted in Figure 7.3. Then one must consider the boundary terms in the augmented Lagrangian formulation (7.3). This section covers a family of related approaches—which we collectively refer to as "Nitsche's method"—for formally eliminating the Lagrange multiplier from (7.3). These methods avoid the inf-sup difficulties associated with discretizing the saddle-point problem.

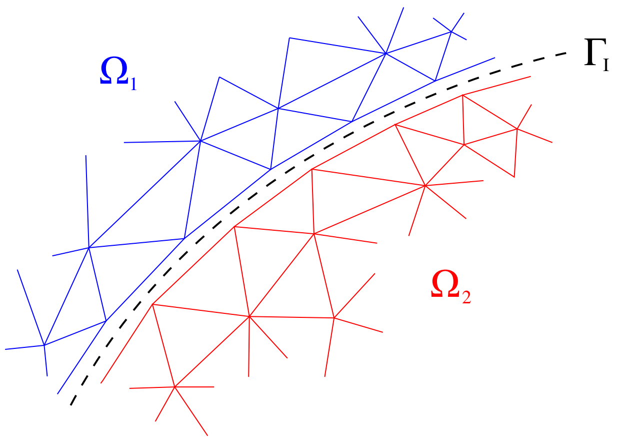

Figure 7.3: Illustration of non-matching FSI, with fluid and structure meshes separated slightly from the interface Γ_I for visibility.

We begin in Section 7.4.1, by introducing a classic idea for enforcing Dirichlet boundary conditions from the 1970s, in the context of the scalar Poisson equation. Section 7.4.3 then develops an analogous method starting from the formulation of Section 7.2, and compares it with this classic approach.

7.4.1 Nitsche's method for the Poisson equation

This section expounds on residual-based stabilization of interface Lagrange multipliers [79]. In particular, we summarize the analysis by Stenberg [80], which ultimately recovers the now-famous method of Nitsche [81], originally derived from other considerations. For simplicity, we consider the model problem of using Lagrange multipliers to enforce Dirichlet boundary conditions on the scalar Poisson equation. The generalization of Nitsche's method to enforcement of the FSI kinematic constraint is treated less rigorously in Section 7.4.3.

We first formulate the Poisson problem on domain Ω with a Dirichlet boundary condition using Lagrange multipliers to apply the boundary condition as a constraint (rather than building it into the trial space): Find $u \in H^1(\Omega)$ and $\lambda \in H^{-1/2}(\Gamma)$ such that $\forall v \in H^1(\Omega)$ and $\mu \in H^{-1/2}(\Gamma)$

$$(\nabla u, \nabla v) + \langle \lambda, v \rangle + \langle \mu, u - g \rangle = (f, v)\ , \tag{7.8}$$

where $\Gamma = \partial\Omega$, $g \in H^{1/2}(\Gamma)$ is the prescribed Dirichlet data, $f \in L^2(\Omega)$ is a source term, and $(\cdot,\cdot)$ is the $L^2(\Omega)$ inner product. We use the notation $\langle \cdot,\cdot \rangle$ to denote ${}_{H^{-1/2}(\Gamma)}\langle \cdot,\cdot \rangle_{H^{1/2}(\Gamma)}$, as can be inferred from the above instance. However, it may be understood as $\langle \cdot,\cdot \rangle = (\cdot,\cdot)_{L^2(\Gamma)}$ in discrete spaces, where $\langle \tilde{F}, \cdot \rangle$ with functional $\tilde{F} \in H^{-1/2}(\Gamma)$ can be represented by $(F,\cdot)_{L^2(\Gamma)}$, with function $F \in L^2(\Gamma)$. (And, as with regular distributions in Section 1.5.2, we shall abuse notation by using the same symbol for F and $\tilde{F}$.)

Remark 57. This formulation can be derived from the constrained optimization problem of minimizing $\frac{1}{2}|u|^2_{H^1(\Omega)} - (f, u)$ subject to the constraint that $\mathrm{tr}_\Gamma\, u = g$. In the Lagrange multiplier formalism, one would seek a stationary point of the functional

$$J(u,\lambda) := \frac{1}{2}|u|^2_{H^1(\Omega)} - (f,u) + \langle \lambda, u - g\rangle \ . \tag{7.9}$$

It is easy to see that the condition $D_{\{v,\mu\}}J(u,\lambda) = 0$ for arbitrary $\{v,\mu\}$ leads to the weak problem (7.8); cf. the unconstrained case explored in Exercise 8 of Chapter 1.

The problem (7.8) clearly has a saddle-point structure, and thus Bubnov–Galerkin discretizations of it will require careful selection of of an inf-sup stable pair of finite-dimensional subsets $V^h \subset H^1(\Omega)$ and $\Lambda^h \subset H^{-1/2}(\Gamma)$. To avoid this difficulty, we try the same trick used in Section 5.4, namely, introducing a term based on a residual of the strong problem to obtain some coercivity in the Lagrange multiplier. In the present problem, our residual involving λ for strong solutions can be easily derived using integration by parts (cf. (7.5)–(7.6)):

$$\lambda + \nabla u \cdot \mathbf{n} = 0 \ , \tag{7.10}$$

where $\mathbf{n}$ is the outward-facing unit normal to Γ. We therefore consider the discrete formulation: Find $u^h \in V^h$ and $\lambda^h \in \Lambda^h$ such that, $\forall v^h \in V^h$ and $\mu^h \in \Lambda^h$

$$\left(\nabla u^h, \nabla v^h\right) + \left\langle \lambda^h, v^h\right\rangle + \left\langle \mu^h, u^h - g\right\rangle - \alpha h\left\langle \lambda^h + \nabla u^h\cdot\mathbf{n}, \mu^h + \nabla v^h\cdot\mathbf{n}\right\rangle = \left(f, v^h\right) \ , \tag{7.11}$$

where $\alpha > 0$ is a stabilization constant (to be determined through subsequent analysis) and h is the element length scale for quasi-uniform meshes used to define finite element spaces V^h and Λ^h. This stabilized formulation is known as the **Barbosa–Hughes method**, after the authors of [79].

Continuing our analogy to PSPG, we introduce mesh-dependent norms in which to carry out error analysis. In particular, let

$$\|v\|^2_{1/2,h} := h^{-1}\|v\|^2_{L^2(\Gamma)} \ , \tag{7.12}$$

for $v \in H^1(\Omega)$, and

$$\|z\|^2_{-1/2,h} := h\|z\|^2_{L^2(\Gamma)} \ , \tag{7.13}$$

for $z \in L^2(\Gamma)$. Next, use these to define

$$\|v\|_{1,h} := |v|_{H^1(\Omega)} + \|v\|_{1/2,h} \tag{7.14}$$

and

$$|||v,\mu||| := \|v\|_{1,h} + \|\mu\|_{-1/2,h} \ , \tag{7.15}$$

where (7.15) is the norm in which we will conduct error analysis. The suitability of this norm depends on a series of properties that are intuitive (and proven in detail by [82]). First, it is clear, from definitions, that

$$\langle v, z\rangle \leq \|v\|_{1/2,h}\|z\|_{-1/2,h} \ . \tag{7.16}$$

Further, we have useful interpolation estimates in the mesh dependent norms:

$$\inf_{v^h\in V^h}\left\|v^h - u\right\|_{1,h} \leq Ch^k\|u\|_{H^{k+1}(\Omega)} \quad \forall u \in H^{k+1}(\Omega) \ , \tag{7.17}$$

and

$$\inf_{\mu^h\in\Lambda^h} \left\|\mu^h - \lambda\right\|_{-1/2,h} \le Ch^{l+3/2}\|\lambda\|_{H^{l+1}(\Gamma)} \quad \forall\lambda \in H^{l+1}(\Gamma)\,, \tag{7.18}$$

which imply

$$\inf_{\{v^h,\mu^h\}\in V^h\times\Lambda^h} \left|\left|\left|v^h - u, \mu^h - \lambda\right|\right|\right| \le C\left(h^k\|u\|_{H^{k+1}(\Omega)} + h^{l+3/2}\|\lambda\|_{H^{l+1}(\Gamma)}\right)\,, \tag{7.19}$$

assuming that $V^h = \mathrm{CG}^h_k$ on Ω and $\Lambda^h = \mathrm{DG}^h_l$ on Γ. Lastly, there is a trace-inverse estimate bounding the boundary flux in terms of the interior solution:

$$C_I\left\|\nabla v^h\cdot\mathbf{n}\right\|^2_{-1/2,h} \le \left|v^h\right|^2_{H^1(\Omega)} \quad \forall v^h \in V^h\,, \tag{7.20}$$

where $C_I \ge 0$ is independent of h. It is also easy to see that the bilinear form

$$B_h(\{u,\lambda\},\{v,\mu\}) := (\nabla u,\nabla v) + \langle\lambda, v\rangle + \langle\mu, u\rangle - \alpha h\langle\lambda + \nabla u\cdot\mathbf{n}, \mu + \nabla v\cdot\mathbf{n}\rangle \tag{7.21}$$

of the problem (7.11) is bounded with respect to the norm $|||\cdot|||$.

What remains, then, is to show inf-sup stability of B_h with respect to $|||\cdot|||$. First, we use the trace-inverse estimate (7.20) to show that, for arbitrary $\{v,\mu\}\in V^h\times\Lambda^h$,

$$\begin{aligned}
B_h(\{v,\mu\},\{v,-\mu\}) &= \|v\|^2_{H^1(\Omega)} + \alpha h\left(\|\mu\|^2_{L^2(\Gamma)} - \|\nabla v\cdot\mathbf{n}\|^2_{L^2(\Gamma)}\right) && (7.22)\\
&\ge (1-\alpha C_I^{-1})|v|^2_{H^1(\Omega)} + \alpha h\|\mu\|^2_{L^2(\Gamma)} && (7.23)\\
&\ge C_1\left(|v|^2_{H^1(\Omega)} + \|\mu\|^2_{-1/2,h}\right)\,, && (7.24)
\end{aligned}$$

under the assumption that $0 < \alpha < C_I$, where C_1 is some positive constant independent of h. Designating

$$\bar\mu := h^{-1}\Pi^h v \quad\Rightarrow\quad \|\bar\mu\|_{-1/2,h} = \left\|\Pi^h v\right\|_{1/2,h}\,, \tag{7.25}$$

where Π^h is the $L^2(\Gamma)$ projector onto Λ^h, we have

$$\begin{aligned}
B_h(\{v,\mu\},\{0,\bar\mu\}) &= \langle v,\bar\mu\rangle - \alpha h\langle\mu + \nabla v\cdot\mathbf{n},\bar\mu\rangle && (7.26)\\
&= h^{-1}\left\|\Pi^h v\right\|^2_{L^2(\Gamma)} - \alpha\left\langle\mu + \nabla v\cdot\mathbf{n}, \Pi^h v\right\rangle && (7.27)\\
&\ge \left\|\Pi^h v\right\|^2_{1/2,h} - (\|\nabla v\cdot\mathbf{n}\|_{-1/2,h} + \|\mu\|_{-1/2,h})\left\|\Pi^h v\right\|_{1/2,h} && (7.28)\\
&\ge \left\|\Pi^h v\right\|^2_{1/2,h} - C_2\left(|v|_{H^1(\Omega)} + \|\mu\|_{-1/2,h}\right)\left\|\Pi^h v\right\|_{1/2,h} && (7.29)\\
&\ge \left\|\Pi^h v\right\|^2_{1/2,h} - \frac{1}{2}\left(C_2^2\left(|v|_{H^1(\Omega)} + \|\mu\|_{-1/2,h}\right)^2 + \left\|\Pi^h v\right\|^2_{1/2,h}\right) && (7.30)\\
&\ge \frac{1}{2}\left\|\Pi^h v\right\|^2_{1/2,h} - C_3\left(|v|^2_{H^1(\Omega)} + \|\mu\|^2_{-1/2,h}\right)\,, && (7.31)
\end{aligned}$$

where C_2 and C_3 are positive constants independent of h. Defining $\{z,\eta\} = \{v, -\mu + \delta\bar\mu\}$, with $\delta < C_1/C_3$, and using the results (7.24) and (7.31),

$$\begin{aligned}
B_h(\{v,\mu\},\{z,\eta\}) &= B_h(\{v,\mu\},\{v,-\mu\}) + \delta B_h(\{v,\mu\},\{0,\bar\mu\}) && (7.32)\\
&\ge (C_1 - \delta C_3)|v|^2_{H^1(\Omega)} + \frac{\delta}{2}\left\|\Pi^h v\right\|^2_{1/2,h} + (C_1 - \delta C_3)\|\mu\|^2_{-1/2,h} && (7.33)\\
&\ge C|||v,\mu|||^2\,, && (7.34)
\end{aligned}$$

where, in the last step, we have used (7.18) to control the projection error $\|\Pi^h v - v\|_{1/2,h}$ (assuming that h is sufficiently small), then used equivalence (in the sense of (1.6)) between Euclidean and taxicab norms. Noting that (7.25) implies

$$\begin{aligned} |||z, \eta||| &= \|v\|_{1,h} + \| - \mu + \delta\bar{\mu}\|_{-1/2,h} && (7.35) \\ &\leq \|v\|_{1,h} + \|\mu\|_{-1/2,h} + \delta\|\bar{\mu}\|_{-1/2,h} && (7.36) \\ &\leq \|v\|_{1,h} + \|\mu\|_{-1/2,h} + \delta\left\|\Pi^h v\right\|_{1/2,h} && (7.37) \\ &\leq C|||v, \mu||| \,, && (7.38) \end{aligned}$$

and combining this with (7.34), we get inf-sup stability of the form B_h with respect to the norm $|||\cdot|||$:

$$\inf_{v,\mu} \sup_{z,\eta} \frac{B_h(\{v, \mu\}, \{z, \eta\})}{|||v, \mu|||\,|||z, \eta|||} \geq C \,, \tag{7.39}$$

where C is independent of h, implying convergence at the rate given by the interpolation estimate (7.19).

As with PSPG, the inf-sup stability of this stabilized method is independent of the choices of spaces. However, unlike with PSPG, we can take $\Lambda^h = L^2(\Gamma)$ and formally eliminate λ^h and μ^h from the problem, posing a discrete problem over V^h only. In particular, when $\Lambda^h = L^2(\Gamma)$, we can re-write (7.11) as: Find $u^h \in V^h$ such that $\forall v^h \in V^h$

$$\left(\nabla u^h, \nabla v^h\right) - \left\langle \nabla u^h \cdot \mathbf{n}, v^h \right\rangle - \left\langle \nabla v^h \cdot \mathbf{n}, u^h - g \right\rangle + (\alpha h)^{-1} \left\langle u^h - g, v^h \right\rangle = \left(f, v^h\right) \,. \tag{7.40}$$

This is known as **Nitsche's method**, due to it having been discovered earlier by Nitsche [81], following a different line of reasoning. A more direct analysis of Nitsche's method, without appeal to Barbosa–Hughes stabilization, is given at the end of the article by Stenberg [80] on which this section is based. Stenberg wrote, in 1995, that "Unfortunately, this [Nitsche's] method seems to be quite unknown", despite its appealing simplicity and efficiency. However, it has since become famous in the PDE literature, and is by far the most popular variant of the Barbosa–Hughes scheme in practical problems. The next section discusses an extension of Nitsche's method to the more complicated setting of FSI, along with a different (albeit heuristic) derivation of it, starting from the augmented Lagrangian formulation of Section 7.2.

Remark 58. A different, heuristic derivation of Nitsche's method is to add a penalty term of the form $\beta\langle u - g, v\rangle$ to the left-hand side of the original problem (7.8), to obtain an augmented Lagrangian formulation, then make the substitutions

$$\lambda \;\rightarrow\; \gamma(u) \cdot \mathbf{n} \quad \text{and} \quad \mu \;\rightarrow\; D_v\gamma(u) \cdot \mathbf{n} \,, \tag{7.41}$$

where $\gamma(u) = -\nabla u$ is the flux and $D_v\gamma(u) = -\nabla v$. This provides a template to generalize Nitsche's method to other PDE systems involving a flux divergence. However, as we show in Section 7.4.3, a practical extrapolation of Nitsche's method to a different problem may require further adjustments, informed by numerical analysis considerations.

Remark 59. The third term of (7.40) clearly still maintains strong consistency if its sign is flipped, because it is based on the residual of the boundary condition. In that case, one can see that the corresponding method is coercive without the fourth penalty term, so α can be arbitrarily large (i.e., the penalty of $u^h - g$ can go to zero). This results in the parameter-free **non-symmetric Nitsche method**, eliminating the question of how best to choose α or define h, which may become unclear in some cases [83]. However, the method (7.40) has generally been favored in the literature, due to its symmetric bilinear form and better provable L^2 convergence rate.

A complete FEniCS implementation of Nitsche's method is given in the following code listing, using the method of manufactured solution (cf. Exercise 2 in Chapter 3) to check accuracy:

```
from dolfin import *

# Number of elements across domain:
N = 32
# Polynomial degree of FE space:
k = 1
# Scaling factor on penalty; can be zero with gamma = -1:
alpha = Constant(1e1)
# 1 for symmetric, -1 for non-symmetric:
gamma = Constant(1)
# 1 for Nitsche method, 0 for weakly-consistent penalty:
nitsche = Constant(1)

# Mesh and function space setup:
mesh = UnitSquareMesh(N,N)
n = FacetNormal(mesh)
h = CellDiameter(mesh)
x = SpatialCoordinate(mesh)
V = FunctionSpace(mesh,"CG",k)

# Choose exact solution for testing:
g = sin(x[0])*exp(x[1])
f = -div(grad(g))

# Pose formulation and solve:
u = TrialFunction(V)
v = TestFunction(V)
res_interior = (dot(grad(u),grad(v)) - f*v)*dx
res_bdry = (nitsche*(-dot(grad(u),n)*v
                     - gamma*dot(grad(v),n)*(u-g))
            + (alpha/h)*(u-g)*v)*ds
res = res_interior + res_bdry
u = Function(V)
solve(lhs(res)==rhs(res),u)

# Check error:
e = g - u
print("L^2 error = "+str(sqrt(assemble(e*e*dx))))
print("H^1 error = "
      +str(sqrt(assemble(dot(grad(e),grad(e))*dx))))
```

The convergence properties of the basic Nitsche method, its non-symmetric variant discussed in Remark 59, and the basic weakly-consistent penalty method can be easily explored by modifying the appropriately labeled parameters in this code.

7.4.2 Nitsche's method for advection–diffusion

We now consider an extension of Nitsche's method to the steady advection–diffusion problem. In particular, we consider the problem: Find $u : \Omega \to \mathbb{R}$ such that

$$\begin{cases} -\nabla \cdot (\kappa \nabla u) + \mathbf{a} \cdot \nabla u = f & , \quad \text{in } \Omega \\ u = g & , \quad \text{on } \partial\Omega \end{cases} , \tag{7.42}$$

where κ, $\mathbf{a}$, and f have the same interpretations as in (6.1), and the function $g : \partial\Omega \to \mathbb{R}$ is given Dirichlet boundary data. A naive attempt to formally extrapolate from the Poisson equation, based on applying the heuristic of Remark 58 to the diffusive flux $\gamma(u) = -\kappa\nabla u$, yields the following discrete problem: Find $u^h \in V^h$ such that $\forall v^h \in V^h$,

$$\begin{aligned} &\left(\kappa\nabla u^h, \nabla v^h\right) + \left(\mathbf{a} \cdot \nabla u^h, v^h\right) - \left\langle \kappa\nabla u^h \cdot \mathbf{n}, v^h\right\rangle - \left\langle \kappa\nabla v^h \cdot \mathbf{n}, u^h - g\right\rangle \\ &\quad + \kappa(\alpha h)^{-1}\left\langle u^h - g, v^h\right\rangle = \left(f, v^h\right) . \end{aligned} \tag{7.43}$$

However, the term containing $\mathbf{a}$ no longer contributes neutral coercivity as it did with strongly-enforced Dirichlet boundary conditions (cf. Exercise 9 in Chapter 1). However, the following strongly-consistent boundary term can be added to the left-hand side of (7.43), to cancel the potentially non-coercive contribution:

$$-\left\langle \{\mathbf{a} \cdot \mathbf{n}\}_- \left(u^h - g\right), v^h\right\rangle , \tag{7.44}$$

where $\{\cdot\}_-$ takes the negative part of its argument, i.e.,

$$\{f\}_- := \frac{1}{2}\left(f - |f|\right) . \tag{7.45}$$

The consistency of (7.44) is clear from its linearity in the boundary condition residual $u^h - g$. The factor $\{\mathbf{a} \cdot \mathbf{n}\}_-$ means that (7.44) is only active on the "inflow part" of the boundary, where the advection velocity points into the domain.

Remark 60. If the advection term of (7.43) is integrated by parts, the sum of the resulting new boundary term and (7.44) will be

$$\left\langle \{\mathbf{a} \cdot \mathbf{n}\}_- g + \{\mathbf{a} \cdot \mathbf{n}\}_+ u^h, v^h\right\rangle , \tag{7.46}$$

where $\{\cdot\}_+$ isolates the positive part of its argument. In this form, it is clear that (7.44) corresponds to a choice of **upwind numerical flux** for advection: the advective flux is computed using g on the inflow part of the boundary and using the solution u^h on the outflow part of the boundary. Comparing with the derivation of a basic DG method in Section 2.5.2, we see that Nitsche's method is essentially using "DG at the boundary", to consistently and stably handle the discontinuity between the discrete solution and the Dirichlet boundary data.

This method can be implemented using FEniCS for the 1D advection–diffusion model problem of Section 6.4 by appending the following code to the script given in that section:

```
# Function satifying BCs:
g = x
# Penalty constant; must be sufficiently
# large for stability.
C_pen = Constant(1e1*k*k)
# Boundary terms of residual:
n = FacetNormal(mesh)
res_weak = (-kappa*dot(grad(u),n)*v
            - kappa*dot(grad(v),n)*(u-g)
            + kappa*(C_pen/h)*(u-g)*v
            - Min(dot(a,n),Constant(0))*(u-g)*v)*ds
res += res_weak

# No strongly-enforced Dirichlet BCs:
uh_weak = Function(V)
solve(lhs(res)==rhs(res),uh_weak)
```

Plotting the resulting solution `uh_weak` and comparing with the solution `uh` computed previously using strong boundary conditions, one can see that when $ah/\kappa \gg 1$, there is a large error in the boundary condition residual, $(u^h - g)|_{x=1}$. However, the solution using weak boundary condition enforcement remains much closer to the exact solution in the region excluding the exact solution's boundary layer. In this sense, Nitsche's method can be seen as striking a balance between error in boundary condition enforcement and error in the interior of the domain, while retaining optimal convergence behavior. Remark 62 discusses the implications of this property for fluid and FSI analysis.

7.4.3 Weak imposition of FSI kinematics

With a rigorous understanding of Nitsche's method in hand, we now return to the FSI problem formulated in Section 7.2, and consider a more heuristic development of an analogous method for FSI coupling suitable for non-matching fluid and structure meshes, following the reasoning first presented in [74].

We start by returning to the traction compatibility equation (7.6), which is analogous to (7.10) in the Poisson model problem. Using this, we can see that, for a solution satisfying the kinematic constraint,

$$\boldsymbol{\lambda} = -\alpha\boldsymbol{\sigma}_1 \cdot \mathbf{n}_1 + (1-\alpha)\boldsymbol{\sigma}_2 \cdot \mathbf{n}_2 \, , \tag{7.47}$$

for any scalar α. In a wide range of engineering applications, the constitutive model for Cauchy stress in terms of fluid velocity is a simple Newtonian viscosity law (4.40), which is much simpler and less likely to be altered than the model for Cauchy stress in terms of the solid's displacement. Thus, it is often simpler to consider the case of $\alpha = 1$, which isolates the fluid's Cauchy stress. Further, in many aerospace and civil engineering applications, the structure is much heavier than the fluid it interacts with, and FSI solution strategies based on the fluid Cauchy stress often perform well in that setting, for reasons that will be discussed in Chapter 9.

Rather than passing through a Barbosa–Hughes-type formulation, then choosing an infinite-dimensional space for $\boldsymbol{\lambda}$, we proceed directly to a Nitsche-type method for the case of a Newtonian incompressible fluid. In particular, we eliminate the Lagrange multiplier with the substitutions

$$\boldsymbol{\lambda} \quad\rightarrow\quad -(2\mu\nabla^s\mathbf{v}_1 - p\mathbf{I})\cdot\mathbf{n}_1 \qquad\text{and}\qquad \delta\boldsymbol{\lambda} \quad\rightarrow\quad -(2\mu\nabla^s\mathbf{w}_1 + q\mathbf{I})\cdot\mathbf{n}_1\ . \tag{7.48}$$

The sign of the q term may at first be counter-intuitive when reasoning by formal analogy to Nitsche's method for the Poisson problem (cf. Remark 58). However, the motivation for it is clear when considering its effect on the coercivity of the bilinear form of a linearized incompressible flow problem (e.g., the Stokes flow model problem considered in Chapter 5). Again following the analogy to Nitsche's method, the penalty parameter β from the augmented Lagrangian formulation is selected to be

$$\beta \sim \mu/h\ , \tag{7.49}$$

where h is a length scale associated with the spatial discretization, e.g., the diameter of elements adjacent to Γ_{I}.

Remark 61. One may make the alternative substitution of

$$\delta\boldsymbol{\lambda} \quad\rightarrow\quad -(-2\mu\nabla^s\mathbf{w}_1 + q\mathbf{I})\cdot\mathbf{n}_1\ , \tag{7.50}$$

which is analogous to the non-symmetric Nitsche variant introduced in Remark 59. In this case, β may be taken to zero while retaining stability.

Another feature of the formulation of [74] that does not follow directly from an analogy to Nitsche's method is an additional term

$$-\int_{\Gamma_{\mathrm{I}}} \rho_1\,\{(\mathbf{v}_1 - \hat{\mathbf{v}})\cdot\mathbf{n}_1\}_-\,(\mathbf{v}_1 - \mathbf{v}_2)\cdot\mathbf{w}_1\,d\Gamma_{\mathrm{I}} \tag{7.51}$$

appearing in the nonlinear residual of the coupled problem, where ρ_1 is the fluid mass density. This term may be understood by analogy to the term (7.44), which we justified through stability analysis in Section 7.4.2's extension of Nitsche's method to advection–diffusion.[2] One can clearly see that the term (7.51) is strongly consistent with the kinematic constraint, as it goes to zero when $\mathbf{v}_1 = \mathbf{v}_2$.

After making all of the formal substitutions and additions mentioned in this section, (7.3) becomes: Find $\{\mathbf{v}_1, p\} \in V_1 \times Q$ and $\mathbf{u}_2 \in V_2$ such that, $\forall\{\mathbf{w}_1, q\} \in V_1 \times Q$ and $\forall\mathbf{w}_2 \in V_2$,

$$\begin{aligned}
&N_1(\{\mathbf{v}_1, p\}, \{\mathbf{w}_1, q\}) + N_2(\mathbf{u}_2, \mathbf{w}_2)\\
&\quad - \int_{\Gamma_{\mathrm{I}}} ((2\mu\nabla^s\mathbf{v}_1 - p\mathbf{I})\cdot\mathbf{n}_1)\cdot(\mathbf{w}_1 - \mathbf{w}_2)\,d\Gamma_{\mathrm{I}}\\
&\quad - \int_{\Gamma_{\mathrm{I}}} ((2\mu\nabla^s\mathbf{w}_1 + q\mathbf{I})\cdot\mathbf{n}_1)\cdot(\mathbf{v}_1 - \mathbf{v}_2)\,d\Gamma_{\mathrm{I}}\\
&\quad - \int_{\Gamma_{\mathrm{I}}} \rho_1\,\{(\mathbf{v}_1 - \hat{\mathbf{v}})\cdot\mathbf{n}_1\}_-\,(\mathbf{v}_1 - \mathbf{v}_2)\cdot\mathbf{w}_1\,d\Gamma_{\mathrm{I}}\\
&\quad + \int_{\Gamma_{\mathrm{I}}} \beta\,(\mathbf{v}_1 - \mathbf{v}_2)\cdot(\mathbf{w}_1 - \mathbf{w}_2)\,d\Gamma_{\mathrm{I}} = 0\ ,
\end{aligned} \tag{7.52}$$

[2]The form of the term (7.51) assumes that advection in the Navier–Stokes weak formulation is treated as in (6.56). While widely-used in practice, this is not the only valid option; alternative weak forms of the advection term imply different boundary terms in place of (7.51), as is clear from the discussion of scalar advection–diffusion in [84].

which involves only the fluid velocity and structure displacement as unknowns. In a practical discrete formulation, the form N_1 would typically be replaced by a stabilized formulation, to account for incompressibility (cf. Chapter 5) and/or advection (cf. Chapter 6), but this is orthogonal to the treatment of the kinematic constraint at the fluid–structure interface.

Remark 62. One may walk away from this section with the impression that Nitsche's and related weak enforcement methods are a concession, sacrificing enforcement of Dirichlet conditions for extra flexibility in construction of finite element meshes. However, as alluded earlier in Remark 52, **weak enforcement of Dirichlet boundary conditions in fluid dynamics is often (much) more accurate than strong enforcement** when considering the overall solution quality rather than just the residual of the boundary condition. The reason for this is that weak enforcement allows for flow to slip along boundaries at coarse mesh resolutions, effectively serving as a wall model at large Reynolds numbers (taking the element size as a length scale). The analogy between weak boundary conditions and wall models is discussed further in [61]. However, unlike classical wall models, strongly-consistent weak enforcement remains optimally-accurate in the limit of direct numerical simulation, where viscous length scales are fully resolved.

Chapter 8

Coupling between meshes: Immersed boundary methods

8.1 Introduction

This chapter concerns the class of numerical methods for partial differential equations (PDEs) known as **immersed boundary methods**, often shortened to **immersed methods**. One point to clarify at the outset is the various ways in which this term may be interpreted. In the present discussion, we consider the defining trait of "immersed (boundary) methods" to be that boundaries of the PDE domain (or sub-domains of a coupled problem) may cut through element interiors. This is perhaps the broadest interpretation and covers everything that one might see referred to as an immersed method (assuming one reinterprets finite difference and volume methods as special cases of finite element methods).

The reason one must clarify the definition of "immersed boundary" or "immersed" methods is that various authors have proposed different and contradictory taxonomies for this class of methods. A common convention (which we *do not* follow here) is to use the term "immersed boundary method" to refer specifically to the numerical method introduced by C. S. Peskin in 1972 [85], which was the first such approach to become widely-known. Following that convention, one must introduce additional terms to classify other methods, such as "fictitious domain", "immersed interface", "cut cell", "interface capturing", "non-boundary-fitted", etc., the interpretations of which do not appear to be uniform in the literature.

Although immersed methods can—and have—been applied to a wide variety of PDE systems, they are frequently associated with computational fluid dynamics (CFD) and fluid–structure interaction (FSI). This association is both for historical and practical reasons, which are intertwined. The early work by Peskin was focused on FSI of heart valves, which are thin, flexible structures in the hearts of most large animals that passively undergo large deformations in response to blood flow, to permit flow in one direction and cut off flow in the other direction. The extreme deformations of the fluid subproblem domain make the conforming FSI methods covered in earlier lectures difficult to apply. It is much simpler to construct a single function space to be used with different boundary configurations at different times, using an immersed method.

It is often taken as obvious that immersed methods are less accurate than conforming ones, although this is not necessarily the case. Many widely-used immersed methods are indeed low-order, but this is not a fundamental limitation. One can attain optimal-order convergence with immersed methods, albeit at the cost of increased complexity and decreased robustness. This chapter begins by considering one paradigm for high-order immersed boundary analysis (Section 8.2). We then point out some practical difficulties with this and discuss some examples of low-order methods (Section 8.3). Lastly, the extensions of these methods to FSI are discussed (Section 8.4).

Remark 63. This chapter is *not* intended to provide a literature review on immersed methods. We instead emphasize a few illustrative examples to permit deeper technical discussion. Some review articles on immersed methods include [86, 87, 88, 89].

8.2 Quadrature-based high-order immersed methods

This section provides an example of how to attain high-order accuracy with immersed methods. This is not the only way to do so, but one of the most versatile and widely-studied. The central insight is: For optimal convergence rates, **it is sufficient for quadrature to conform to the PDE domain, even if function spaces do not.**

This is not surprising from approximation considerations. For domains with sufficiently-smooth boundaries, there exist smooth extensions of smooth functions to larger domains. Suppose we pose a PDE system on $\Omega \subset \mathbb{R}^d$, but construct a finite element space V_E^h with element size h on Ω_E such that $\Omega \subset \Omega_E$. If the exact PDE system solution $u : \Omega \to \mathbb{R}^n$ can be extended smoothly to $u_E : \Omega_E \to \mathbb{R}^n$, then, based on standard interpolation lemmas, we can find $w_E^h \in V_E^h$ that converges optimally to u_E in an appropriate Sobolev norm as $h \to 0$. This implies that $w^h := w_E^h|_\Omega$ will converge at the same rate to u in that norm. Recalling the basic error analysis of Galerkin's method, we can expect a stable and consistent discrete formulation to deliver a solution $u^h \in V^h := V_E^h|_\Omega$ that converges at least as quickly as w^h.

The main practical challenge is computing $\int_\Omega$ in the weak formulation, when Ω is no longer a union of finite elements, which can each be integrated over using Gaussian quadrature. Some approaches from the literature are:

- Subdivide each element Ω^e cut by $\partial\Omega$ into a conforming mesh of the region $\Omega^e \cap \Omega$, where this mesh is only used for quadrature, and does not add new degrees of freedom or need to satisfy the same compatibility conditions as a mesh used to define a function space. This is easier than constructing a good-quality conforming mesh of Ω, but is still subject to many pathological corner cases. This is the approach taken by the **MultiMesh functionality in FEniCS** [90], but, due to its technical complexity, this code never reached a high level of maturity, and is not currently planned to be maintained in FEniCSx [27].

- Define a non-conforming mesh for cut-cell quadrature, which simply uses a high resolution near the immersed boundary. This is more computationally-expensive than conforming re-parameterization of cut cells (for the same level of accuracy), but it is simpler and more robust. Using the so-called **finite cell method (FCM)** [91, 92], one can recursively refine the quadrature mesh around an immersed boundary, given only a geometry kernel that can say whether a point is inside or outside of Ω. An open-source implementation of the FCM is FCMLab [93].

Another practical challenge is to impose boundary conditions on $\partial\Omega$. Strong imposition of Dirichlet boundary conditions becomes impossible (without brute-force low-order approaches, such as setting to zero all degrees of freedom associated with cut cells), and Neumann boundary conditions or weakly-enforced Dirichlet boundary conditions (e.g., Nitsche's method) require quadrature for the integral $\int_{\partial\Omega}$ when $\partial\Omega$ is no longer a union of element facets. In principle, a quadrature mesh would need to be constructed for $\partial\Omega$ as well, although, in practice, studies from the FCM literature indicate that accuracy of boundary integrals is of relatively less importance than accuracy of volume integrals [94].

8.3 Low-order immersed methods

Given the substantial body of published work on high-order immersed methods, it may at first seem that low-order methods are deficient or obsolete. However, when weighing the total cost for different methods (in terms of both labor and computation) to reach a sufficient level of approximation error in practically-relevant quantities, low-order numerical methods often remain the rational choice. If one demands a sufficiently-small level of numerical error, a higher-order method will always win. However, in many problems of practical interest, the error inherent in modeling a real physical system as a PDE may be orders of magnitude larger than the error level needed to justify the extra complexity of high-order methods.

As a model problem to illustrate issues associated with low-order immersed methods, let us consider the problem of solving a second-order elliptic PDE on some domain $\Omega \subset \mathbb{R}^d$ with the (trace of the) solution u constrained to equal given data g on a surface $\Gamma \subset \overline{\Omega}$, as illustrated in Figure 8.1. If Γ divides Ω into subdomains, as shown in the figure, an equivalent formulation would be to pose two standard elliptic boundary-value problems on these subdomains. However, this becomes difficult in more general cases. The weak form for this problem, using a Lagrange multiplier field on Γ to enforce $\mathrm{tr}_\Gamma\, u = g$, is: Find $(u, \lambda) \in H^1_0(\Omega) \times H^{-1/2}(\Gamma)$ such that $\forall (v, \rho) \in H^1_0(\Omega) \times H^{-1/2}(\Gamma)$,

$$a(u, v) + \langle \lambda, \mathrm{tr}_\Gamma\, v \rangle - \langle \rho, \mathrm{tr}_\Gamma\, u - g \rangle = F(v)\ , \tag{8.1}$$

where $a : H^1_0(\Omega) \times H^1_0(\Omega) \to \mathbb{R}$ is a coervive and bounded bilinear form and $F : H^1_0(\Omega) \to \mathbb{R}$ is linear and bounded. The choice of space for the Lagrange multiplier and its corresponding test function is due to the fact that the trace operator maps $H^1(\Omega)$ functions into $H^{1/2}(\Gamma)$.[1]

Now consider discretizations of the problem (8.1) where the discrete space for approximations of u is an $H^1_0(\Omega)$-conforming finite element space, defined on a mesh that is *not* fitted to the immersed surface Γ. This raises two general problems:

- **Consistency:** The exact solution will, in general, have jumps in ∇u across Γ. This means that u cannot be in $H^2(\Omega)$, so, even using the lowest-order $H^1_0(\Omega)$-conforming CG_1 space to approximate it, the rate of convergence of interpolation error will be controlled by the low regularity of u.

- **Stability:** Obviously (8.1) has the structure of a saddle-point problem, so, to apply the Bubnov–Galerkin method, one would need to identify an inf-sup stable pair of discrete

[1]When Γ intersects $\partial\Omega$, the trace operator actually maps from $H^1_0(\Omega)$ into a slightly smaller space on Γ, whose dual is then slightly larger, but the details are not important for this discussion.

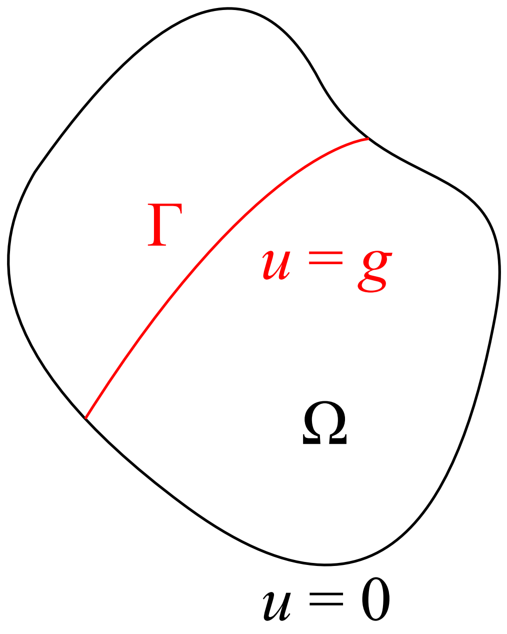

Figure 8.1: Example configuration for the immersed-boundary model problem used to discuss low-order methods.

> spaces for the solution and Lagrange multiplier. We will see that attempts to circumvent this difficulty via Barbosa–Hughes or Nitsche methods will lead back to a consistency error.

We now go into more detail on these issues.

8.3.1 Low-order interpolation error

Recall the basic interpolation estimate (2.32) for CG_k finite element spaces. If we want $H^1(\Omega)$ interpolation error, then $s = 1$. We also know that r must be less than 2 if ∇u is discontinuous across Γ. Thus, $\alpha = r - 1$ for $k \geq 1$. If we restrict r to integer values, then this tells us that $\alpha \leq 0$, i.e., it is impossible for approximations to converge to the exact solution! This conundrum leads us to revisit fractional-order Sobolev spaces in more detail, to understand the convergence behavior of low-order immersed boundary methods.

The first question to ask is: What does it mean to raise the derivative operator to a non-integer power? There are a number of characterizations of fractional derivatives in the mathematical literature, but perhaps the most intuitive one relies on the Fourier transform. Consider the 1D case, for simplicity. Given $f : \mathbb{R} \to \mathbb{R}$, recall that

$$\mathcal{F}\left(\left(\frac{d}{dx}\right)^{\ell} f\right)(\xi) = (i\xi)^{\ell}\mathcal{F}(f)(\xi) \tag{8.2}$$

where $\mathcal{F}$ is the Fourier transform, given by

$$\mathcal{F}(f)(\xi) = \frac{1}{\sqrt{2\pi}}\int_{\mathbb{R}} f(x)e^{-i\xi x}\,dx \tag{8.3}$$

for $f \in L^1(\mathbb{R})$, which can be extended to an invertible mapping between more general spaces by first restricting to a class of smooth functions, then applying density arguments. Thus

$$\left(\frac{d}{dx}\right)^{\ell} f = \mathcal{F}^{-1}\left((i\xi)^{\ell}\mathcal{F}(f)\right) , \tag{8.4}$$

where the right-hand side above is meaningful for non-integer ℓ. This motivates an alternate definition of Hilbert–Sobolev spaces:

$$H^{\ell}(\mathbb{R}) = \left\{f \in L^2(\mathbb{R}) \ : \ \mathcal{F}^{-1}\left(\left(1 + |\xi|^2\right)^{\ell/2}\mathcal{F}f\right) \in L^2(\mathbb{R})\right\} . \tag{8.5}$$

Based on the above discussion of fractional differential operators, this corresponds to requiring square-integrability of

$$\left(1 - \left(\frac{d}{dx}\right)^2\right)^{\ell/2} f , \tag{8.6}$$

where the differential operator is of leading order ℓ. One can prove that this definition of H^{ℓ} is equivalent to that of Section 1.6 when ℓ is an integer.

If we consider a 1D instance of our model problem (8.1), Γ will be a point, x_0, in the interval Ω, and there may be a jump in du/dx at x_0. The question we are then interested in answering is: For what ℓ does $H^{\ell}(\mathbb{R})$ contain a function $f : \mathbb{R} \to \mathbb{R}$ that is smooth aside from a jump discontinuity at x_0? Let us compute:

$$\sqrt{2\pi}\,(\mathcal{F}f)(\xi) = \int_{-\infty}^{\infty} f(x)e^{-i\xi x}\,dx \tag{8.7}$$

$$= \int_{-\infty}^{x_0} f(x)e^{-i\xi x}\,dx + \int_{x_0}^{\infty} f(x)e^{-i\xi x}\,dx . \tag{8.8}$$

Integrating by parts,

$$\sqrt{2\pi}\,(\mathcal{F}f)(\xi) = (i\xi)^{-1}\left(\int_{-\infty}^{x_0} f'(x)e^{-i\xi x}\,dx + \int_{x_0}^{\infty} f'(x)e^{-i\xi x}\,dx\right) + (i\xi)^{-1}\,e^{-i\xi x_0}\left(f(x_0^-) - f(x_0^+)\right) . \tag{8.9}$$

Assuming f decays rapidly to zero at infinity, the Riemann–Lebesgue lemma implies that the integral terms converge to zero as $|\xi| \to \infty$, so, in the high-frequency limit, (8.9) decays like $|\xi|^{-1}$. Thus, to have

$$(1 + |\xi|^2)^{\ell/2}\mathcal{F}f \in L^2(\mathbb{R}) , \tag{8.10}$$

we need we need the high-frequency tails of

$$(1 + |\xi|^2)^{\ell/2}|\xi|^{-1} \tag{8.11}$$

to be square-integrable. This requires

$$2(2(\ell/2) - 1) = 2(\ell - 1) < -1 \qquad \Longleftrightarrow \qquad \ell < 1/2 , \tag{8.12}$$

since we know that $\int_{\xi_0}^{\infty} \xi^{-p}\, d\xi$ diverges for $p \leq 1$. Similarly, for a function f with a jump in f', like the solution to our model immersed boundary problem, we can only assume that f is in $H^{3/2-\epsilon}(\mathbb{R})$, for $\epsilon > 0$. Going back to the interpolation estimate (2.32), which remains valid for fractional-order spaces, we can take $r = 3/2 - \epsilon$, and expect a convergence rate of $H^1(\Omega)$ interpolation error that is arbitrarily close to $1/2$.

Remark 64. A sanity check to improve one's intuition about fractional convergence rates is to consider a CG_1 interpolant of the Heaviside function on the interval $(-1, 1)$, using a uniform mesh with an even number of elements. In this case, it is easy to directly calculate that the L^2 norm of the interpolation error is $O(h^{1/2})$.

However, one frequently hears that low-order immersed methods are "first-order", not "half-order". This is due to the tacit choice of a different norm for measuring error. We have already seen numerically, in some earlier homework exercises, that, although we only proved optimal convergence in one norm, we get higher convergence rates in weaker norms. For instance, when solving the Poisson equation with sufficiently-regular problem data, one obtains a convergence rate of k in H^1, and a convergence rate of $k+1$ in L^2. Returning to our immersed model problem, if we consider interpolation error in L^2, i.e., $s = 0$, the rate of convergence of interpolation error would be $3/2 - \epsilon$. However, this is *not* what is obtained in numerical solutions. To understand why, we must look in more detail at the arguments used to show higher convergence rates in weaker norms.

The simplest and best-known proof of a higher convergence rate in a weaker norm is known in the literature as the **Aubin–Nitsche trick**. To illustrate this trick, consider the weak problem: Find $u \in V = H^1_0(\Omega)$ such that $\forall v \in V$,

$$a(u, v) = L(v)\ , \tag{8.13}$$

where a and L satisfy the hypotheses of the Lax–Milgram theorem. Let $u^h \in V^h = \mathrm{CG}^h_k$ be the solution to a Bubnov–Galerkin approximation, and define the error $e = u - u^h$. The **dual problem** is: Find $\phi \in V$ such that $\forall v \in V$,

$$a(v, \phi) = (e, v)_{L^2(\Omega)}\ . \tag{8.14}$$

(Notice that the solution ϕ is the *second* argument to a.) Choosing $v = e$ and using Galerkin orthogonality, we can introduce an arbitrary $w^h \in V^h$:

$$a(e, \phi - w^h) = \|e\|^2_{L^2(\Omega)}\ . \tag{8.15}$$

If w^h solves the dual a-projection problem

$$a(v^h, w^h - \phi) = 0 \quad \forall v^h \in V^h\ , \tag{8.16}$$

then we can expect $\|w^h - \phi\|_{H^1}$ to converge at the same rate as error in interpolation of ϕ:

$$\|w^h - \phi\|_{H^1} \leq C h^{\min\{k, r-1\}} \|\phi\|_{H^r}\ , \tag{8.17}$$

and, for $r \leq 2$, elliptic regularity theory (with some mild assumptions on a) gives us $\|\phi\|_{H^r} \leq C\|e\|_{L^2}$. Further, we always take $k \geq 1$, so the second branch of the min is always taken, and

$$\|e\|^2_{L^2} = a(e, \phi - w^h) \leq C\|e\|_{H^1}\|\phi - w^h\|_{H^1} \leq C\|e\|_{L^2} h^{\alpha + r - 1}\ , \tag{8.18}$$

where α is the convergence rate of e in H^1.

The interpretation of this is that the boost in convergence rate of $||e||_{L^2}$ relative to $||e||_{H^1}$ depends on how large we can take r. If the form a is "nice" (i.e., has smooth coefficients), then we can choose $r = 2$, but, if a induces lower regularity, then r will be reduced. In particular, if a induces a jump in $\nabla\phi$, e.g., through a discontinuous coefficient, or a reaction term concentrated on an immersed surface, we would be limited to $r < 3/2$. The form a would also induce a jump in ∇u, leading to $\alpha = 1/2 - \epsilon$, so $||e||_{L^2}$ would converge at a rate of $1 - \epsilon$. This is the sense in which immersed methods that approximate discontinuous derivatives on unfitted meshes are "first-order".

Remark 65. If reduced regularity in u is due solely to the source functional L, e.g., a is nice, but L includes a singular distribution in $H^{-1}(\Omega)$, this does not affect the regularity of the dual solution ϕ, and the $L^2(\Omega)$ convergence rate would be $\alpha + 1$. One might naively attempt to draw an analogy between the preceding Aubin–Nitsche argument and the elliptic model problem (8.1) by considering the term $\langle\lambda, v\rangle$ to be a concentrated source term within L in the problem (8.13). However, λ is a part of the unknown solution to (8.1), and appears in the bilinear form.

Remark 66. The Aubin–Nitsche trick is closely related to proofs of **local error estimates** using the Nitsche–Schatz theory [95], where errors in regions separated from irregular solution features may converge faster than the global error. Nitsche and Schatz use a duality argument similar to the Aubin–Nitsche trick, which allows one to show optimal convergence away from singular source terms, but not when irregularities are built into the bilinear form.

Remark 67. A clever way of exploiting the Nitsche–Schatz theory in a partially-immersed setting is the so-called **fat boundary method** [96].

8.3.2 Stability issues

As mentioned earlier, consistency is not the only difficulty plaguing immersed discretizations of our model problem. There is no obvious way to ensure uniform inf-sup stability of a Galerkin discretization for all different ways that Γ might cut through the mesh defining V^h. One's first instinct might be to attempt to apply the Barbosa–Hughes Petrov–Galerkin method discussed in previous lectures, or its special case, Nitsche's method, in which the Lagrange multiplier is formally eliminated. However, the strong problem residual that would be used in the Petrov–Galerkin approach would, for our model problem, be that λ equals the *jump* in flux across Γ. When Γ cuts through element interiors, the discrete solution gradient ∇u^h is continuous across Γ, and the jump in flux is zero. In that case, Barbosa–Hughes stabilization and Nitsche's method both reduce to simpler weakly-consistent methods.

Remark 68. If g is assumed to satisfy some problem posed on the surface Γ, then the residual of this surface problem can be instead used to formally eliminate λ. This would be like choosing $\alpha = 0$ instead of $\alpha = 1$ in (7.47), and is loosely analogous to Peskin's original immersed boundary method, where the elastic internal forces of a structure transported by the fluid velocity are used to calculate a source term in the fluid subproblem.

Let us go into more detail about the degenerated version of Nitsche's method, which becomes a straightforward penalty approach: Find $u^h \in V^h$ such that, for all $v^h \in V^h$,

$$a(u^h, v^h) + \beta(u^h - g, v^h)_{L^2(\Gamma)} = F(v^h)\,, \tag{8.19}$$

where $\beta \sim 1/h$. Even if Γ coincides with $\partial\Omega$, and we no longer have the fundamental interpolation issues discussed in Section 8.3.1, this would converge with only first-order accuracy in $L^2(\Omega)$, regardless of the choice of k in the finite element space used to approximate u. This can be demonstrated easily by modifying the Nitsche's method implementation in Section 7.4.1. The method (8.19) is an example of a weakly-consistent method in which the consistency error introduced by the formulation is, at worst, comparable to the unavoidable interpolation error.

Remark 69. The formulation (8.19) can also be obtained directly, without passing through Barbosa–Hughes/Nitsche, by introducing the approximation

$$\lambda \approx \beta(u - g) . \tag{8.20}$$

This is a more standard approach to deriving penalty regularizations of constraints.

8.4 Immersed CFD and FSI

We now briefly survey some examples from the literature of how the types of immersed-boundary methods discussed above can be applied to the CFD and FSI problems they are most closely associated with. We focus on applications grounded in variational methods, which correspond most directly to the preceding discussion. With some additional effort, one can extend that correspondence to finite difference and/or volume methods (cf. Sections 2.5.1 and 2.5.2), in which terms much of the published literature on immersed CFD and FSI is written. However, a full literature review is outside the scope of this section, and we refer the interested reader to the review articles cited in Section 8.1.

8.4.1 Finite cell CFD and FSI

Although the use of immersed methods is frequently justified by appeal to large motions of boundaries, they may also alleviate challenges in constructing meshes for fixed-boundary CFD. This is illustrated by an industrial-scale example in Figure 8.2, which contrasts fitted and unfitted meshes for simulating flow around the complex geometry of an agricultural tractor, as published in [97]. The general approach for constructing the quadrature rule used for the unfitted analysis is illustrated schematically in Figure 8.3. It is clear that this construction relies only on a single geometrical query: Is a point $\mathbf{x}$ inside or outside of the fluid domain Ω_1? A general-purpose approach to doing this for closed immersed surfaces is via ray-tracing. A ray emanating from a point will intersect a closed surface an odd number of times if the point is inside and an even number of times if the point is outside. Ray–surface intersection algorithms have already been heavily optimized by the computer graphics community, albeit also tailored for a different setting. Detailed investigations of the accuracy of finite cell CFD for engineering quantities of interest can be found in [97, 98].

Extending the finite cell method to boundary movement, as needed for FSI, brings with it the challenge of being able to very rapidly build cut-cell quadrature rules in each time step of a computation. The bridge deck simulations of [99] are a pioneering example of 3D finite cell FSI. The recent study [100] goes into great detail about efficient inside/outside tests for building adaptive quadrature rules, which combine ray tracing with other techniques.

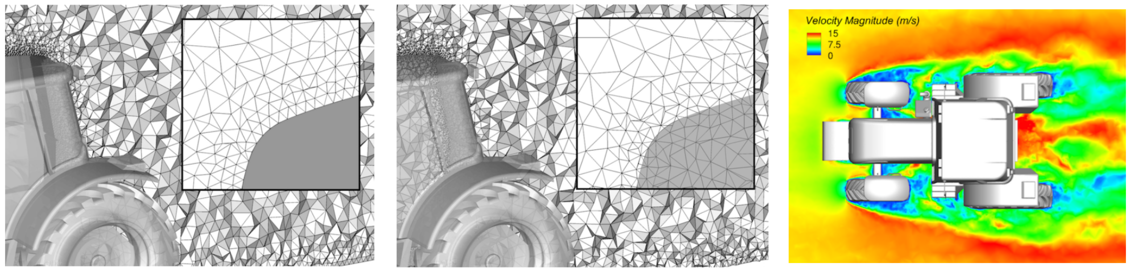

Figure 8.2: Figures adapted from [97], comparing conforming (left) and immersed (middle) meshes for flow (right) around a complex geometrical model of an agricultural tractor.

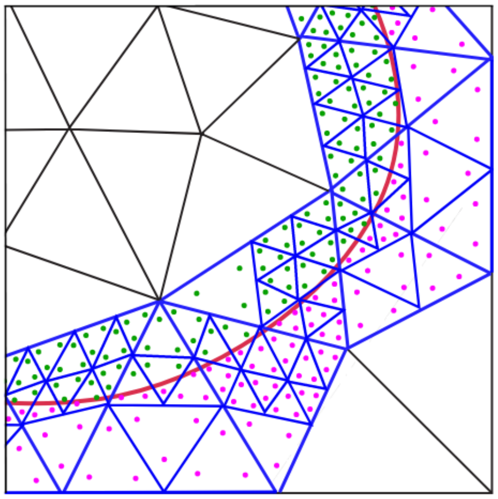

Figure 8.3: Schematic 2D illustration, adapted from [97], of a finite cell quadrature scheme. Elements (black) are cut by an immersed boundary (red), and quadrature points inside the problem domain (pink) are used to integrate a variational problem. Recursive subdivision (blue) of cut elements is for *quadrature only*, and does not add new degrees of freedom.

8.4.2 Low-order immersed FSI with thin structures

Testing which element of a mesh a point is in and what its parametric coordinates are in that element is a substantially-easier geometric query than the inside/outside test needed to construct quadrature rules like those shown in Figure 8.3. The fact that this simpler query is all that is needed to implement numerical quadrature of the same order of accuracy as the interpolation error inherent to low-order immersed methods (cf. Section 8.3.1) is a significant practical advantage of low-order immersed methods. This advantage is especially pronounced if the fluid mesh has some structure that may be exploited, e.g., a tensor-product construction. Although not couched in the variational language used here, the original immersed FSI calculations of Peskin [85] were arguably an instance of this, with the Lagrangian markers following the structure acting as a quadrature rule.

A pioneering example of low-order immersed FSI analysis based on variational principles (drawing on concepts introduced earlier by Glowinski et al. [101]) is the work by Baaijens, de Hart, and collaborators in the early 2000s [102, 103], which coupled a finite element discretization of thin, deforming heart valve leaflets to a non-matching finite element discretization of incompressible flow, into which the leaflets were immersed. Ultimately, this forward-looking research proved to be too far ahead of its contemporary computer technology and scientific software ecosystem to develop into a practical simulation technology for heart valves. However, more recent efforts to simulate heart valve FSI using variational methods [104, 105, 106, 46, 107] have found more success in jumping from the numerical methods literature into biomedical research [108]. Figure 8.4 qualitatively illustrates the effectiveness of low-order immersed FSI methods for reproducing structural deformations of an experimental model of a heart valve. Reference [109] analyzes the formulation of this later work using a scalar model problem like that of Section 8.3, and an openly-available FEniCS-based implementation is described in [110].

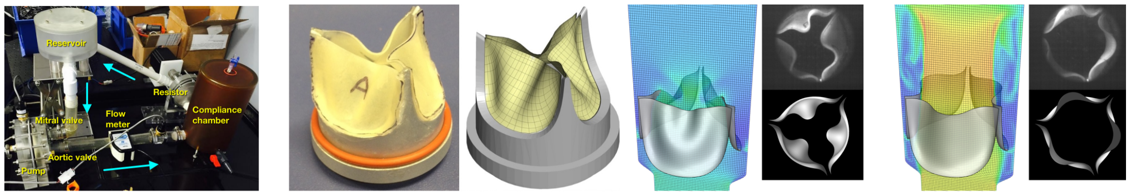

Figure 8.4: Comparison of low-order immersed FSI simulations with *in vitro* experiments, adapted from [46]. Taking structural displacements as a quantity of interest, error between the computations and PDE system is likely already smaller than errors between the PDE system and physical reality, so the efficiency and robustness of low-order methods outweigh their inaccuracy.

Chapter 9

Coupling between solvers: Staggered solution schemes

9.1 Introduction

It is often the case that one wants to use existing computational implementations of solvers for the individual subproblems of a coupled system. As such, it may be convenient to solve these subproblems separately. However, can these independent solutions be carried out sequentially in some iterative algorithm—i.e., a **subiterative solution scheme (or subiteration)**—that converges to the desired coupled solution? This is not a trivial question, and a great deal of research has been done on it, often in the context of specific application domains.

One of the most common analytical tools for characterizing the convergence behavior of iterative solution methods is the Banach contraction mapping theorem, which we will cover in Section 9.2. After that, Section 9.3 applies this theorem to a highly-simplified model which illustrates some of the issues faced by subiterative solution schemes in the context of fluid–structure interaction (FSI).

9.2 The Banach contraction mapping theorem

Consider an iterative algorithm of the form $x_{n+1} = T(x_n)$, where $\{x_n\} \subset X$ are a sequence of (hopefully convergent) guesses for the solution in some non-empty complete metric space X, with metric d, and $T : X \to X$ is a mapping from X to itself. The mapping T is considered a **contraction mapping** if $\exists q \in [0, 1)$ such that $\forall x, y \in X$,

$$d(T(x), T(y)) \leq q d(x, y) , \tag{9.1}$$

i.e., the images of points under T are closer together than the points themselves. The **Banach contraction mapping theorem** states that, if T is a contraction mapping, then the sequence $\{x_n\}$ converges to some $x \in X$.

The proof of this theorem is remarkably simple. Let the starting point of the sequence, $x_0 \in X$, be arbitrary. Then the definition of a contraction mapping obviously implies

$$d(x_{n+1}, x_n) \leq q^n d_0 , \tag{9.2}$$

where $d_0 := d(x_0, x_1)$. Using this, we can also show that $\{x_n\}$ is Cauchy. Taking $m > n > N > 0$,

$$\begin{aligned} d(x_m, x_n) &\leq d(x_m, x_{m-1}) + \cdots + d(x_{n+1}, x_n) && (9.3)\\ &\leq \left(q^{m-1} + \cdots + q^n\right) d_0 && (9.4)\\ &\leq \left(q^n \sum_{j=0}^{\infty} q^j\right) d_0 && (9.5)\\ &= \left(\frac{q^n}{1-q}\right) d_0 && (9.6)\\ &\leq \left(\frac{q^N}{1-q}\right) d_0 \,. && (9.7) \end{aligned}$$

Because $0 \leq q < 1$, it obtains that $\forall \epsilon > 0$, $\exists N$ such that,

$$q^N \leq \epsilon \frac{1-q}{d_0} \,, \tag{9.8}$$

where we disregard the case of $d_0 = 0$ as trivial. Thus, $\forall \epsilon > 0$, $\exists N$ such that, $\forall m, n > N$,

$$d(x_m, x_n) \leq \epsilon \,, \tag{9.9}$$

i.e., $\{x_n\}$ is Cauchy. Because we assumed X is complete, this implies that $x_n \to x \in X$.

The limit x is also a **fixed point** of the mapping T, i.e.,

$$T(x) = x \,, \tag{9.10}$$

which can be shown by commuting T with the limit $n \to \infty$. (This commutation is allowable because T is continuous, which in turn follows from it being a contraction mapping.) Further, a fixed point of T must be unique, since the distance between images of distinct fixed points would not be able to decrease, as required of a contraction mapping. These easy corollaries are often included in the contraction mapping theorem, which is then referred to as the **Banach fixed-point theorem**.

Remark 70. The constructive nature of the proof also provides information about how fast the sequence $\{x_n\}$ converges to x:

$$d(x_n, x) \leq \frac{q^n}{1-q} d_0 \,, \tag{9.11}$$

i.e., the convergence is more rapid as $q \to 0$ and stagnates as $q \to 1$.

9.3 The added mass effect in FSI

When a structure is immersed in a fluid, its interactions with the fluid can cause it to behave qualitatively as if it is more massive than it would be in isolation. If a subiterative solution scheme solves the structure subproblem independently of the fluid subproblem, then this "added mass" is not accounted for, which often adversely affects the stability of the solution scheme. To illustrate this as simply as possible, we introduce a model problem of two masses, m_1 and m_2, connected by a rigid rod, and constrained to move in 1D.[1] In the context of FSI, this may be considered to behave like a piston of mass m_2 interacting with an incompressible fluid of mass m_1. Its equations of motion are:

$$m_1 \dot{v}_1 = \lambda \tag{9.12}$$
$$m_2 \dot{v}_2 = -\lambda \tag{9.13}$$
$$v_1 = v_2 \ , \tag{9.14}$$

where v_i is the velocity of mass i and λ is a Lagrange multiplier to enforce the third constraint equation. Discretizing implicitly in time, using the backward Euler method for simplicity, we have, for mass 1,

$$m_1 \left(\frac{v_1 - v_1^{\text{old}}}{\Delta t} \right) = \lambda \ , \tag{9.15}$$

where superscripts indicate time level and Δt is the time step size, and, likewise for mass 2,

$$m_2 \left(\frac{v_2 - v_2^{\text{old}}}{\Delta t} \right) = -\lambda \ . \tag{9.16}$$

We now consider how to approximate the solution (v_1, v_2, λ) as the (hopefully extant) limit of a sequence $\{(v_1^j, v_2^j, \lambda^j)\}$ generated by some iterative solution procedure, with iterations indexed by j. Obviously a Newton iteration on the entire coupled problem would converge in a single step, because the problem is linear. However, if we are committed to solving subproblems 1 and 2 individually, this model may still be instructive.

9.3.1 The classical Dirichlet-to-Neumann iteration

The **Dirichlet-to-Neumann** FSI coupling algorithm is to apply the structure velocity as a Dirichlet boundary condition on the fluid, solve the fluid subproblem, extract the traction on the fluid–structure interface, apply this as a Neumann boundary condition on the structure, and repeat. In the context of our model problem, the steps in each iteration are:

1. $v_1^{j+1} = v_2^j$, i.e., apply a Dirichlet condition to subproblem 1.
2. Solve (9.15) for λ^{j+1}, i.e., extract the traction from subproblem 1.
3. Solve (9.16) for v_2^{j+1}, i.e., solve subproblem 2 with a Neumann condition.
4. $j \leftarrow j + 1$.

We can see the conditions for convergence of this iteration by looking at the expression for v_2^{j+1} and recalling the contraction mapping theorem:

$$v_2^{j+1} = -\frac{m_1}{m_2} v_2^j + \frac{m_1 v_1^{\text{old}} + m_2 v_2^{\text{old}}}{m_2} \ . \tag{9.17}$$

[1]Richer model problems are studied in [111, 112], but lead to similar practical conclusions.

Clearly the mapping from v_2^j to v_2^{j+1} is only a contraction mapping if $m_1 < m_2$. In the context of FSI, this means that the fluid is light and the structure is heavy. An application where subiteration is likely to be appropriate is aeroelastic flutter in aircraft wings or bridge decks, where the structure is orders of magnitude denser than the surrounding fluid. An application where subiteration is likely to be unstable is cardiovascular FSI, where thin structures interact with a fluid of similar density.

Remark 71. Note that the "mass of fluid the structure interacts with" is a somewhat imprecise quantity outside of this simplified 1D piston-like example. Further, in the case of a *compressible* fluid, this mass depends on the time step size, i.e., the structure only interacts with fluid that is within $\sim c\Delta t$ of it, where c is the sound speed. In that case, reducing the time step is a viable approach to improving the convergence rate. The incompressible case corresponds to the limit of $c \to \infty$, and reducing the time step *does not* improve convergence when the structure is light relative to the fluid. These issues are discussed with a richer model problem, in more detail, by van Brummelen [112].

Remark 72. It is possible to stabilize Dirichlet-to-Neumann subiteration by scaling-down the update to v_2 within each pass of the iteration, although attempting to remedy a large m_1/m_2 ratio in this manner may result in very slow convergence. Researchers have nonetheless had found some success in applying *ad hoc* extensions of Aitken acceleration to update the solution update scaling factor within FSI subiteration [113].

Remark 73. As elaborated in [112], the stability of subiteration is closely related to the stability of (the most basic variants of) **staggered time integration**, where fluid and structure subproblems are each solved only once within a time step.

9.3.2 Penalty coupling

We now consider the case of this model problem for a fluid and structure coupled by penalty forces, as discussed for low-order immersed boundary methods in Section 8.3.2. In that case, we make the approximation

$$\lambda \approx \beta(v_2 - v_1) , \tag{9.18}$$

where $\beta > 0$ is the penalty parameter. In our model problem, the time-discrete equations become

$$m_1\left(\frac{v_1 - v_1^{\text{old}}}{\Delta t}\right) = \beta(v_2 - v_1) , \tag{9.19}$$

and

$$m_2\left(\frac{v_2 - v_2^{\text{old}}}{\Delta t}\right) = \beta(v_1 - v_2) . \tag{9.20}$$

A subiterative solution scheme to this penalty-coupled problem would perform the following steps in each iteration:

1. Solve (9.19) for v_1^{j+1}, holding v_2 fixed at v_2^j.
2. Solve (9.20) for v_2^{j+1}, holding v_1 fixed at v_1^{j+1}, as obtained in the previous step.
3. $j \leftarrow j + 1$.

As in the Dirichlet-to-Neumann case, we can express this as a mapping from v_2^j to v_2^{j+1}:

$$v_2^{j+1} = -\frac{\beta^2 v_2^j}{(\beta + m_1/\Delta t)(\beta + m_2/\Delta t)} + \frac{\beta m_1 v_1^{\text{old}}/\Delta t}{(\beta + m_1/\Delta t)(\beta + m_2/\Delta t)} + \frac{m_2 v_2^{\text{old}}/\Delta t}{\beta + m_2/\Delta t} \ . \tag{9.21}$$

This is clearly a contraction mapping with

$$q = \frac{\beta^2}{\left(\beta + \frac{m_1}{\Delta t}\right)\left(\beta + \frac{m_2}{\Delta t}\right)} < 1 \ . \tag{9.22}$$

However, we can see that $q \to 1$ as $\beta \to \infty$. Thus, recalling (9.11), the more strongly we attempt to enforce the constraint, the more slowly this iteration will converge (although it is very robust, in the sense that q is never > 1, unlike the Dirichlet-to-Neumann iteration).

The result that $\beta \to \infty$ may be troubling in light of the scaling of $\beta = C_1 h^{-1}$, suggested for immersed boundary condition enforcement in Section 8.3.2, through our analogy to Nitsche's method. However, if we take $\Delta t = C_2 h$, i.e., quasi-uniformity in space–time, we have that

$$q \to \frac{C_1^2}{C_1^2 + \frac{m_1 m_2}{C_2^2}} \tag{9.23}$$

as $h \to 0$, by substituting into (9.22), then applying l'Hospital's rule. In other words, q asymptotes to a value strictly less than one under refinement. Thus, we can still expect subiteration to converge reliably under mesh refinement for penalty-based low-order immersed methods, so long as appropriate relationships hold between β, h, and Δt, with moderate constants C_1 and C_2.

Appendix A

Integration by parts

This appendix contains optional exercises to clarify the variety of related integration by parts formulas used throughout distributional calculus, finite element methods, and continuum mechanics. For further review of multivariate calculus, the text of Greenberg [114] is a classic reference.

A.1 The divergence theorem

The calculus identities in question all stem from the **divergence theorem**,

$$\int_{\Omega} \nabla \cdot \mathbf{u} \, d\Omega = \int_{\partial\Omega} \mathbf{u} \cdot \mathbf{n} \, d\Gamma \,, \tag{A.1}$$

where Ω is a volume, $\mathbf{u}$ is a vector field, $\partial\Omega$ is the boundary of Ω, and $\mathbf{n}$ is the outward-facing normal vector to $\partial\Omega$.

1. First consider a simplified version of the theorem. Suppose

   $$\mathbf{u} = u\mathbf{e}_1 \,, \tag{A.2}$$

   where u is a scalar function, i.e., the vector field points only in the x_1 direction. Then the divergence theorem is:

   $$\int_{\Omega} \frac{\partial u}{\partial x_1} \, d\Omega = \int_{\partial\Omega} u\mathbf{n} \cdot \mathbf{e}_1 \, d\Gamma \,. \tag{A.3}$$

   If we assume that proofs for simplified versions with vector fields pointing in the x_2 and x_3 directions are similar, why is this sufficient to prove the general case, (A.1)?

2. In 1D (i.e., Ω is an interval, $a < x_1 < b$), show that (A.3) is just the fundamental theorem of calculus.

3. Extend the result of Exercise 2 to the 2D case, where Ω is an axis-aligned rectangle.

4. Now consider a more general 2D case, where Ω is a 2D area, and Γ is a curve, as shown in Figure A.1. Let us split Γ into two segments, Γ_{left} and Γ_{right}, as shown in Figure A.1. Parameterize each of Γ_{left} and Γ_{right} by x_2. For example, let Γ_{right} be the parametric curve

   $$\mathbf{c}_{\text{right}}(x_2) = g_{\text{right}}(x_2)\mathbf{e}_1 + x_2\mathbf{e}_2 \,, \tag{A.4}$$

where the function g_{right} is as shown in Figure A.1. How would one integrate the arc length of Γ_{left} or Γ_{right}, using this parameterization?

5. What are the outward-facing normal vectors to Γ_{left} and Γ_{left} in terms of (derivatives of) g_{left} and g_{right}?

6. Now consider (A.3). The left side is

$$\int_{x_2^{\text{bottom}}}^{x_2^{\text{top}}} \left(\int_{g_{\text{left}}(x_2)}^{g_{\text{right}}(x_2)} \frac{\partial u}{\partial x_1}\, dx_1 \right) dx_2\, . \tag{A.5}$$

Using the previous answers about the arc lengths and normal vectors of Γ_{left} and Γ_{right}, show that (A.5) simplifies to the right-hand side of the 2D divergence theorem (A.3).

Hint: Use the fundamental theorem of calculus on the inner integral. The "missing" factor of **n**'s x_1 component in the outer integral comes from changing variables from x_2 to arc length.

7. How would one fix this proof when Ω is as shown in Figure A.2?

8. How can this approach be extended to prove the 3D version of the divergence theorem?

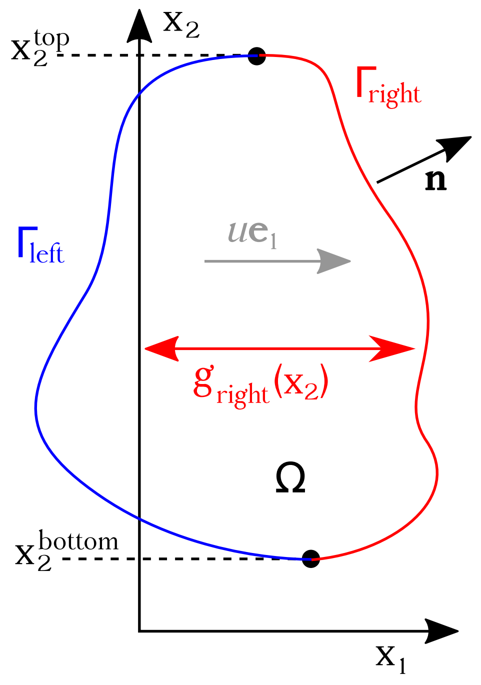

Figure A.1: Notation for use in Exercises 4–6.

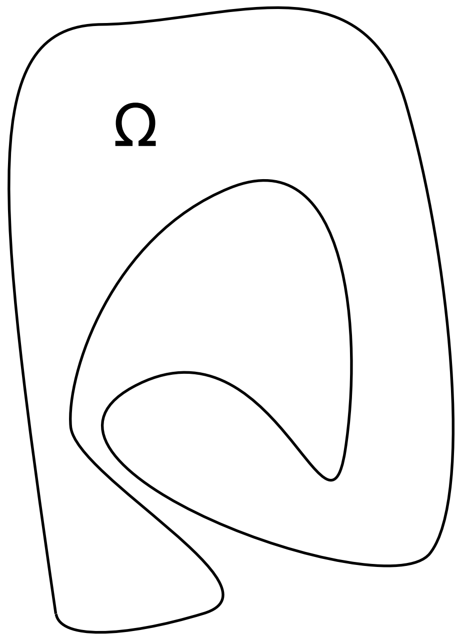

Figure A.2: Less straightforward case, for problem 7.

A.2 Corollaries: Vector and tensor integration by parts

Finite element practitioners commonly refer to a variety of multivariate calculus identities as "**integration by parts**". Here we will illustrate how these identities follow from the divergence theorem.

1. Use the divergence theorem to show that
$$-\int_\Omega v\Delta u \, d\Omega = \int_\Omega \nabla v \cdot \nabla u \, d\Omega - \int_\Gamma v\nabla u \cdot \mathbf{n} \, d\Gamma \,. \tag{A.6}$$
How is this similar to 1D integration by parts?

2. Suppose **u** and **v** are vector-valued functions. Then their gradients are matrix-valued, with entries
$$(\nabla \mathbf{u})_{ij} = \frac{\partial u_i}{\partial x_j} \tag{A.7}$$
and likewise for $\nabla\mathbf{v}$. The divergence of a matrix-valued function is then a vector, with components
$$(\nabla \cdot \mathbf{A})_i = \left(\frac{\partial A_{i1}}{\partial x_1} + \frac{\partial A_{i2}}{\partial x_2} + \frac{\partial A_{i3}}{\partial x_3}\right) = \sum_{j=1}^{3} \frac{\partial A_{ij}}{\partial x_j} \,, \tag{A.8}$$
i.e., we treat the second index like we treat the only index of a vector when taking the divergence.

   (a) Write out a formula for the i^{th} component of the Laplacian of **u**, i.e. $\Delta\mathbf{u} := \nabla \cdot (\nabla\mathbf{u})$.

(b) Use the divergence theorem to show that we get

$$-\int_{\Omega} \mathbf{v} \cdot \Delta \mathbf{u} \, d\Omega = \int_{\Omega} \nabla \mathbf{v} : \nabla \mathbf{u} \, d\Omega - \int_{\Gamma} \mathbf{v} \cdot ((\nabla \mathbf{u})\mathbf{n}) \; d\Gamma \,, \tag{A.9}$$

where $(\nabla \mathbf{u})\mathbf{n}$ is a matrix–vector product,

$$((\nabla \mathbf{u})\mathbf{n})_i = \sum_{j=1}^{3} (\nabla \mathbf{u})_{ij} \, \mathbf{n}_j \,, \tag{A.10}$$

and the colon symbol indicates a matrix dot product,

$$\mathbf{A} : \mathbf{B} = \sum_{i,j=1}^{3} A_{ij} B_{ij} \,. \tag{A.11}$$

Hint: Use a divide-and-conquer strategy; the dot product in the left-hand side of (A.9) can be written as a sum of three terms. Consider just one term from this sum, say, the i^{th} term. This is the product of scalar function v_i and the Laplacian of another scalar function u_i, and one can mostly re-use the argument from Exercise 1, then sum over i at the end.

Appendix B

Tensor calculus notation

This appendix reviews standard notations used in tensor calculus, along with the corresponding functions in FEniCS's UFL, and some common tricks applied in derivations.

B.1 Index notation

Many basic vector equations are expressed in the traditional "direct notation" commonly taught in undergraduate vector calculus courses, where differential operators are written using the ∇ symbol. This has the risk of becoming ambiguous when applied to more complicated tensor equations, though, and it becomes clearer and less error-prone to carry out derivations in **index notation**, where statements about tensor-valued quantities are made about an arbitrary element of their vector, matrix, or higher-dimensional array representation, in some arbitrary, unspecified coordinate chart, e.g.,

$$(\nabla\alpha)_j = \frac{\partial\alpha}{\partial x^j}\ , \tag{B.1}$$

where α is a scalar field, $\{x^j\}_{j=1}^d$ is a coordinate chart on a d-dimensional manifold, and the relation is understood to hold for all $j \in \{1,\ldots,d\}$. The choice of superscript and subscript positions for indices relates to covariance and contravariance of tensors, but we shall not invoke these concepts here. Aside from special topics like general curved shell structures and beams, engineering presentations of continuum mechanics typically restrict index notation to a **global Cartesian coordinate chart**, which always exists when modeling physical space as Euclidean. In that case, the technical distinction between superscript and subscript indices in (B.1) becomes moot, and one would simply write

$$(\nabla\alpha)_j = \frac{\partial\alpha}{\partial x_j}\ . \tag{B.2}$$

Unless otherwise specified, we always follow this convention. Another important implication of this convention for differential operators, as in (B.1) and (B.2), is that, in Cartesian coordinates, basis vectors are constant with respect to coordinates. If α in (B.1) were replaced with a vector field $\mathbf{v} = v^1\mathbf{e}_1 + \cdots + v^d\mathbf{e}_d$, where $\{\mathbf{e}_i\}$ is a covariant basis associated with $\{x^i\}$, formulas for $(\nabla\mathbf{v})^i{}_j$ would included additional terms associated with the partial derivatives of basis vectors (called **Christoffel symbols**). These extra terms are zero and typically omitted in Cartesian index notation. One must therefore remain mindful of the fact that equations expressed in this reduced, subscript-only version of index notation will no longer be true in non-Cartesian coordinate charts, e.g., polar coordinates, even if the manifold is considered to be a Euclidean space. A thorough presentation of elasticity using arbitrary curvilinear coordinate charts on manifolds can be found in [40].

In (B.2), j is a "free index", meaning, as mentioned above, that its value could be anything in the range $\{1, \dots, d\}$. Certain operations involve summing over the range of indices, e.g., application of a linear operator to a vector:

$$(\mathbf{A}\mathbf{v})_i = \sum_{j=1}^{d} A_{ij} v_j \,, \tag{B.3}$$

where $\mathbf{v} \in \mathbb{R}^d$ and $\mathbf{A} : \mathbb{R}^d \to \mathbb{R}^d$ is linear. This relation holds for any $i \in \{1, \dots, d\}$, so i is a free index. The index j, on the other hand, is a **summation index** or **dummy index**. Summations become cumbersome to write, so it is a standard convention to assume that, within a given multiplicative term of any equation, **any repeated index is summed over**. This summation is also commonly referred to as **contraction**, and one would say that the right-hand side of (B.3) "contracts over j".

B.2 Index notation in UFL

FEniCS's UFL provides index notation to express complicated tensor equations. Indices are their own type of algebraic object, and the function `indices` can be imported through the module `ufl` to return a tuple of several indices, e.g.,

```
from ufl import indices
i,j,k,l = indices(4)
```

One may then express a tensor equation like

$$T_{ij} = R_{ikl} S_{jkl} \,, \tag{B.4}$$

where $\mathbf{T}$, $\mathbf{R}$, and $\mathbf{S}$ are rank-3 tensors, as

```
T = as_tensor(R[i,k,l]*S[j,k,l], (i,j))
```

assuming the indices `i,j,k,l` have been declared (as above) and the tensors `R` and `S` are defined elsewhere. Note that a tuple of free indices is passed as a second argument. The reason this is necessary is to set the order of free indices in the left-hand side of (B.4). Scalars without any free indices can also be defined directly, e.g.,

$$\alpha = \mathbf{u} \cdot \mathbf{v} = u_j v_j \tag{B.5}$$

could be written

```
alpha = u[j]*v[j]
```

assuming that the index `j` and vectors `u` and `v` are already defined.

B.3 Unambiguous direct notation in UFL

Despite the clarity of index notation, many people nonetheless find direct notation more aesthetically-appealing. However, for direct notation to be available in UFL, it must be absolutely unambiguous, since statements of equations must compile into unique code for numerical routines. This section provides index notation definitions of some common direct notations used in UFL.

The most commonly-seen vector calculus operations are gradient and divergence. In UFL, `grad` is defined for a tensor `T` of arbitrary rank as

```
S = grad(T)
```

which corresponds in mathematical direct and index notation to

$$\mathbf{S} = \nabla\mathbf{T} \quad \text{and} \quad S_{i\cdots jkl} = \frac{\partial T_{i\cdots jk}}{\partial x_l} , \tag{B.6}$$

where $\{x_j\}$ is a Cartesian coordinate chart on physical space. The divergence of $\mathbf{T}$ contracts over the last two indices of $\mathbf{S}$, with the UFL code

```
R = div(T)
```

corresponding to the mathematical notation

$$\mathbf{R} = \nabla\cdot\mathbf{T} \quad \text{and} \quad R_{i\cdots j} = S_{i\cdots jkk} = \frac{T_{i\cdots jk}}{\partial x_k} . \tag{B.7}$$

An alternative interpretation of the symbol ∇ that is frequently seen in fluid mechanics is defined in UFL as `nabla_grad`. The difference between `grad` and `nabla_grad` is in the order of indices of $\mathbf{S}$ in "$\mathbf{S} = \nabla\mathbf{T}$". When "$\nabla$" is understood as `nabla_grad`, the corresponding index notation is

$$S_{li\cdots k} = \frac{\partial T_{i\cdots k}}{\partial x_l} , \tag{B.8}$$

i.e., the new index from the partial derivative is the *first* index of $\mathbf{S}$ instead of the last.[1] The reason for including this operation is tied to the interpretation of the `dot` operation. The UFL code

```
T = dot(U,V)
```

corresponds to the mathematical notation

$$\mathbf{T} = \mathbf{U}\cdot\mathbf{V} \quad \text{and} \quad T_{i\cdots jk\cdots l} = U_{i\cdots jm}V_{mk\cdots l} , \tag{B.9}$$

i.e., it contracts over the last index of its first operand and first index of its last operand. Thus, the common direct notation from fluid mechanics of

$$\text{“}\, \mathbf{w} = \mathbf{u}\cdot\nabla\mathbf{v} \,\text{”} \tag{B.10}$$

has the desired interpretation of

$$w_i = u_j \frac{\partial v_i}{\partial x_j} \tag{B.11}$$

when ∇ is understood as `nabla_grad`, e.g.,

```
w = dot(u, nabla_grad(v))
```

[1]There is likewise a `nabla_div` function in UFL, which contracts over first two indices of the output of `nabla_grad`, but this function is not frequently used, as it is equivalent to `div` for the most common argument types, namely, vectors and symmetric rank-2 tensors.

However, one will *not* obtain the desired result using `grad` in this case. This example illustrates why index notation should be favored for complicated derivations, as traditional hand-written direct notation does not always have a unique interpretation.

Remark 74. The `grad` and `nabla_grad` interpretations of the direct notation "$\nabla\mathbf{T}$" can be thought of as corresponding to the two common index notation abbreviations for partial derivatives,

$$\frac{\partial T_{i\cdots j}}{\partial x_k} = T_{i\cdots j,k} = \partial_k T_{i\cdots j} \ , \tag{B.12}$$

with `grad` adding an index to the end, like the comma notation, and `nabla_grad` adding an index at the beginning, like the ∂ notation.

A common related confusion in UFL is how `dot` differs from `inner`, since they are equivalent when both arguments are rank-1 tensors (i.e., vectors). The `inner` function contracts over *all* indices of both arguments (which must therefore have the same rank); the UFL code

```
alpha = inner(T,S)
```

corresponds to

$$\alpha = \mathbf{T} : \mathbf{S} = T_{i\cdots j} S_{i\cdots j} \ . \tag{B.13}$$

N.b. that this use of the colon operator is inconsistent with some other uses found in the literature, where it denotes contraction over the last two indices of its first operand and first two indices of its last operand, e.g., the notation "$\varepsilon : C : \varepsilon = \varepsilon_{ij} C_{ijkl} \varepsilon_{kl}$" that is often seen in discussions of linear elasticity. This once again demonstrates the potential ambiguity of direct notation.

Conversely, the `outer` function in UFL implements the **outer product**, sometimes also called a **dyadic** or **tensor product**, which is written in direct and index notation as

$$\mathbf{T} = \mathbf{R} \otimes \mathbf{S} \quad \text{or} \quad \mathbf{T} = \mathbf{RS} \ , \quad \text{and} \quad T_{i\cdots jk\cdots l} = R_{i\cdots j} S_{k\cdots l} \ . \tag{B.14}$$

Correspondingly, one sometimes sees the `grad` interpretation of ∇ written in direct notation as

$$\mathbf{S} = \mathbf{T}\nabla \quad \text{or} \quad \mathbf{S} = \mathbf{T} \otimes \nabla \ , \tag{B.15}$$

which clarifies the index ordering of $\mathbf{S}$. However, this convention is not universal and may lead to confusion about what field is being differentiated if further symbols appear to the right of ∇. For example, if we allow partial derivatives in ∇ to act leftward, one might parse (B.10) as $\mathbf{u} \cdot \nabla\mathbf{v} = (\mathbf{u} \cdot \nabla)\mathbf{v} = (\operatorname{div} \mathbf{u})\mathbf{v}$, for yet another unintended interpretation.

B.4 Multi-index notation

Occasionally, it is convenient to condense statements about sets of partial differential operators using **multi-index notation**. A **multi-index** is a tuple of non-negative integers,

$$\alpha = (\alpha_1, \cdots, \alpha_n) \ , \tag{B.16}$$

and its overall order is the sum of its elements, denoted

$$|\alpha| := \sum_{i=1}^{n} \alpha_n \ . \tag{B.17}$$

The partial differential operator ∂^α then has the interpretation

$$\partial^\alpha f(x_1, \cdots, x_n) = \frac{\partial^{|\alpha|} f}{\partial^{\alpha_1} x_1 \cdots \partial^{\alpha_n} x_n}(x_1, \cdots, x_n) \tag{B.18}$$

when applied to a function f of n variables. The notation

$$\sum_{|\alpha|=m} \partial^\alpha f \tag{B.19}$$

would sum all partial differential operators of order m. For example, if $f : \mathbb{R}^n \to \mathbb{R}$ for $n = 2$ and $m = 2$, then we would have

$$\sum_{|\alpha|=2} \partial^\alpha f = \frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_1 x_2} + \frac{\partial^2 f}{\partial x_2^2} , \tag{B.20}$$

where the three terms in the expanded sum correspond to $\alpha = (2, 0)$, $\alpha = (1, 1)$, and $\alpha = (0, 2)$, respectively, i.e., all possible multi-indices of length $n = 2$ with $|\alpha| = m = 2$.

Remark 75. Multi-index notation is not supported in UFL.

B.5 Some useful identities

This section reviews the index and direct notations for some formulas which are frequently encountered in continuum mechanics.

B.5.1 The chain rule

Perhaps the most commonly-used tool in tensor calculus derivations is the **chain rule**. The common statement for some scalar-valued multivariate function f is

$$\frac{\partial f(x_1, \cdots, x_n)}{\partial y} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_n} \frac{\partial x_n}{\partial y} . \tag{B.21}$$

For a tensor-valued function $\mathbf{T}$ depending on a tensor $\mathbf{S}$ which in turn depends on $\mathbf{U}$, we can apply the scalar chain rule to each component:

$$\frac{\partial T_{i \cdots j}}{\partial U_{k \cdots l}} = \frac{\partial T_{i \cdots j}}{\partial S_{m \cdots n}} \frac{\partial S_{m \cdots n}}{\partial U_{k \cdots l}} . \tag{B.22}$$

In direct notation, this is often written

$$\frac{\partial \mathbf{T}}{\partial \mathbf{U}} = \frac{\partial \mathbf{T}}{\partial \mathbf{S}} : \frac{\partial \mathbf{S}}{\partial \mathbf{U}} , \tag{B.23}$$

especially when $\mathbf{S}$ has rank 2. Note, however, that this use of ":" is not consistent with the definition given by (B.13).

B.5.2 Derivatives of inverses and determinants

It often arises that the inverse of a rank-2 tensor **A** must be differentiated with respect to some other tensor **B** on which **A** depends. If one only has an explicit formula for the derivative of **A** with respect to **B**, one can use the chain rule to reduce the problem to finding the derivative of $\mathbf{A}^{-1}$ with respect to **A**:

$$\frac{\partial A^{-1}_{ij}}{\partial B_{k\cdots l}} = \frac{\partial A^{-1}_{ij}}{\partial A_{mn}} \frac{\partial A_{mn}}{\partial B_{k\cdots l}} \tag{B.24}$$

or, following the notational convention of (B.23) for the interpretation of ":",

$$\frac{\partial \mathbf{A}^{-1}}{\partial \mathbf{B}} = \frac{\partial \mathbf{A}^{-1}}{\partial \mathbf{A}} : \frac{\partial \mathbf{A}}{\partial \mathbf{B}} \ . \tag{B.25}$$

The remaining derivative is [115, (1.253)]

$$\frac{\partial A^{-1}_{ij}}{\partial A_{k\ell}} = -A^{-1}_{ik} A^{-1}_{\ell j} \ , \tag{B.26}$$

which cannot be written clearly in direct notation. Derivatives of $\det \mathbf{A}$ with respect to **B** can be dealt with similarly, but using the identity [115, (1.241)]

$$\frac{\partial \det \mathbf{A}}{\partial A_{ij}} = \det \mathbf{A} A^{-1}_{ji} \ . \tag{B.27}$$

In direct notation, this is frequently abbreviated with the notation

$$\frac{\partial \det \mathbf{A}}{\partial \mathbf{A}} = (\det \mathbf{A})\mathbf{A}^{-T} =: \operatorname{Cof} \mathbf{A} \ , \tag{B.28}$$

where the right-hand side is referred to as the "cofactor matrix" of **A**.

Remark 76. In continuum mechanics derivations, the tensor **A** in question is frequently a deformation gradient, as in the applications of (B.26) and (B.27) in Chapter 4.

Appendix C

Full code for conforming FSI

This appendix contains the full code for the example implementation of conforming FSI discussed in Section 7.3. Please refer to that discussion for detailed explanation of the FSI formulation used, and its correspondence to the general theory in Chapter 4. The basic problem setup here is a cantilever beam, oriented vertically and fixed to the bottom of the domain, interacting with a flow around it, which is driven by a time-periodic Dirichlet boundary condition on fluid velocity on the left side of the domain. The right side of the domain has a stable Neumann boundary condition, which sets to zero the traction on parts of the boundary where flow is leaving and the combined traction and advective flux on parts of the domain where the flow is entering. This is a straightforward extension to Navier–Stokes of the stable Neumann boundary condition for advection–diffusion explained in [84].

```
from dolfin import *
from ufl import indices, Jacobian, Min
from mshr import *

####### Domain and mesh setup #######

# Parameters defining domain geometry:
SOLID_LEFT = 0.45
SOLID_RIGHT = 0.55
SOLID_TOP = 0.5
OMEGA_H = 0.75
OMEGA_W = 1.0
REF_MARGIN = 0.1

# Define the mshr geometrical primitives for this domain:
r_Omega = Rectangle(Point(0,0),Point(OMEGA_W,OMEGA_H))
r_Omega_s = Rectangle(Point(SOLID_LEFT,0),
                      Point(SOLID_RIGHT,SOLID_TOP))

# Enumerate subdomain markers
FLUID_FLAG = 0
SOLID_FLAG = 1
```

```
# Zero is the default flag, and does not need to be
# explicitly set for the fluid subdomain.
r_Omega.set_subdomain(SOLID_FLAG, r_Omega_s)

# Parameters defining refinement level:
N = 70

# Generate mesh of Omega, which will have a fitted
# subdomain for Omega_s.
mesh = generate_mesh(r_Omega, N)

# Mesh-derived quantities:
d = mesh.geometry().dim()
n = FacetNormal(mesh)
I = Identity(d)

# Define subdomains for use in boundary condition definitions:
class Walls(SubDomain):
    def inside(self, x, on_boundary):
        return (on_boundary
                and ((x[1] < DOLFIN_EPS)
                     or (x[1] > (OMEGA_H - DOLFIN_EPS))))
class Inflow(SubDomain):
    def inside(self, x, on_boundary):
        return (on_boundary and (x[0] < DOLFIN_EPS))
class Outflow(SubDomain):
    def inside(self, x, on_boundary):
        return (on_boundary and (x[0] > (OMEGA_W - DOLFIN_EPS)))
class SolidDomainClosure(SubDomain):
    def inside(self, x, on_boundary):
        return (x[0] > SOLID_LEFT - DOLFIN_EPS
                and x[0] < SOLID_RIGHT + DOLFIN_EPS
                and x[1] < SOLID_TOP + DOLFIN_EPS)
class SolidDomainInterior(SubDomain):
    def inside(self, x, on_boundary):
        # (Keep boundary at bottom of domain)
        return (x[0] > SOLID_LEFT + DOLFIN_EPS
                and x[0] < SOLID_RIGHT - DOLFIN_EPS
                and x[1] < SOLID_TOP - DOLFIN_EPS)

# Set up integration measures, with flags to integrate over
# subsets of the domain.
markers = MeshFunction('size_t', mesh, d, mesh.domains())
bdry_markers = MeshFunction('size_t', mesh, d-1, 0)
OUTFLOW_FLAG = 2
```

```
Outflow().mark(bdry_markers,OUTFLOW_FLAG)
dx = dx(metadata={'quadrature_degree': 2},
        subdomain_data=markers)
ds = ds(metadata={'quadrature_degree': 2},
        subdomain_data=bdry_markers)

####### Elements and function spaces #######

# Define function spaces (equal order interpolation):
cell = mesh.ufl_cell()
Ve = VectorElement("CG", cell, 1)
Qe = FiniteElement("CG", cell, 1)
VQe = MixedElement((Ve,Qe))
# Mixed function space for velocity and pressure:
W = FunctionSpace(mesh,VQe)
# Function space for mesh displacement field,
# which will be solved for separately in a
# quasi-direct scheme:
V = FunctionSpace(mesh,Ve)

####### Set up time integration variables #######

TIME_INTERVAL = 1e2
N_STEPS = 2000
Dt = Constant(TIME_INTERVAL/N_STEPS)

# Mesh motion functions:
uhat = Function(V)
uhat_old = Function(V)
du = TestFunction(V)
vhat = (uhat-uhat_old)/Dt

# Fluid--structure functions:
(dv, dp) = TestFunctions(W)
w = Function(W)
v,p = split(w)
w_old = Function(W)
v_old, p_old = split(w_old)
dv_dr = (v - v_old)/Dt
dv_ds = dv_dr # (Only valid in solid)

# This is the displacement field used in the
# solid formulation; notice that it is NOT
# in the space V; it is an algebraic object
# involving the unknown fluid--structure velocity
# field v.
u = uhat_old + Dt*v

# This will need to be updated to match u, for
# purposes of setting the boundary condition
# on the mesh motion subproblem.
u_func = Function(V)

####### Changes of variable #######

# Follow notation from Bazilevs et al., where y is
# the coordinate in the reference domain, x is the
# coordinate in the spatial domain, and X is the
# coordinate in the material domain.  Note that
# the built-in differential operators (e.g., grad,
# div, etc.) and integration measures (e.g., dx, ds,
# etc.) are w.r.t. the reference configuration, y,
# which is the mesh that FEniCS sees.
dX = dx(SOLID_FLAG)
dy = dx
grad_y = grad
grad_X = grad # (Only valid in solid)
y = SpatialCoordinate(mesh)
x = y + uhat
det_dxdy = det(grad_y(x))
def grad_x(f):
    return dot(grad_y(f),inv(grad_y(x)))
def div_x(f): # (For vector-valued f)
    return tr(grad_x(f))
def div_x_tens(f): # (For (rank-2) tensor-valued f)
    i,j = indices(2)
    return as_tensor(grad_x(f)[i,j,j],(i,))

# Note:  Trying to define dx = det_dxdy*dy would
# result in an object of type Form, which could no
# longer be used as an integration measure.
# Thus, integration over the spatial configuration
# is done with det_dxdy*dy directly in the fluid
# formulation.

####### Boundary conditions #######

# BCs for the fluid--structure subproblem:
bc0_fs = DirichletBC(W.sub(0), Constant((0.0,0.0)), Walls())
# Note that "x" in this Expression is really y
# in the kinematics described above, but, because the
# mesh motion problem has a zero Dirichlet BC on the inflow
# boundary, there happens to be no difference.
v_in = Expression("16.0*sin(pi*t)*x[1]*(H - x[1])/(H*H)",
                  t=0.0,H=OMEGA_H, degree=2)
bc1_fs = DirichletBC(W.sub(0).sub(0), v_in, Inflow())
bc2_fs = DirichletBC(W.sub(1), Constant(0),
                     SolidDomainInterior())
bcs_fs = [bc0_fs, bc1_fs, bc2_fs]

# BCs for the mesh motion subproblem:
bc_m_walls = DirichletBC(V.sub(1), Constant(0), Walls())
bc_m_inflow = DirichletBC(V, Constant(d*(0,)), Inflow())
bc_m_outflow = DirichletBC(V, Constant(d*(0,)), Outflow())
bc_m_struct = DirichletBC(V, u_func, SolidDomainClosure())
bcs_m = [bc_m_struct,bc_m_walls,bc_m_inflow,bc_m_outflow]


####### Formulation of mesh motion subproblem #######

# Residual for mesh, which satisfies a fictious elastic problem:
F_m = grad_y(uhat) + I
E_m = 0.5*(F_m.T*F_m - I)
m_jac_stiff_pow = Constant(3)
# Artificially stiffen the mesh where it is getting crushed:
K_m = Constant(1)/pow(det(F_m),m_jac_stiff_pow)
mu_m = Constant(1)/pow(det(F_m),m_jac_stiff_pow)
S_m = K_m*tr(E_m)*I + 2.0*mu_m*(E_m - tr(E_m)*I/3.0)
res_m = (inner(F_m*S_m,grad_y(du)))*dy
Dres_m = derivative(res_m, uhat)

####### Formulation of the solid subproblem #######

# Elastic properties
mu_s = Constant(1e4)
K = Constant(1e4)
rho_s0 = Constant(1)

# Kinematics:
F = grad_X(u) + I
E = 0.5*(F.T*F - I)
S = K*tr(E)*I + 2.0*mu_s*(E - tr(E)*I/3.0)
f_s = Constant(d*(0,))
res_s = rho_s0*inner(dv_ds - f_s,dv)*dX \
        + inner(F*S,grad_X(dv))*dX
```

```
####### Formulation of the fluid subproblem #######

# Galerkin terms:
rho_f = Constant(1)
mu_f = Constant(1e-2)
sigma_f = 2.0*mu_f*sym(grad_x(v)) - p*I
v_adv = v - vhat
DvDt = dv_dr + dot(grad_x(v),v_adv)
resGal_f = (rho_f*dot(DvDt,dv) + inner(sigma_f,grad_x(dv))
            + dp*div_x(v))*det_dxdy*dy(FLUID_FLAG)

# Stabilization:

# Deformed mesh size tensor in the spatial configuration:
dxi_dy = inv(Jacobian(mesh))
dxi_dx = dxi_dy*inv(grad_y(x))
G = (dxi_dx.T)*dxi_dx

# SUPG/PSPG:
resStrong_f = rho_f*DvDt - div_x_tens(sigma_f)
Cinv = Constant(1.0)
tau_M = 1.0/sqrt(((2*rho_f/Dt)**2)
                 + inner(rho_f*v_adv,G*(rho_f*v_adv))
                 + Cinv*(mu_f**2)*inner(G,G))
resSUPG_f = inner(tau_M*resStrong_f,
                  rho_f*dot(grad_x(dv),v_adv)
                  + grad_x(dp))*det_dxdy*dy(FLUID_FLAG)
# LSIC/grad-div:
tau_C = 1.0/(tr(G)*tau_M)
resLSIC_f = tau_C*div_x(v)*div_x(dv)*det_dxdy*dy(FLUID_FLAG)

# Stable Neumann BC term, assuming advective
# form of material time derivative term:
v_adv_minus = Min(dot(v_adv,n),Constant(0))
resOutflow_f = -dot(rho_f*v_adv_minus*dv,v)*ds(OUTFLOW_FLAG)

# Note: On a general deforming mesh, the boundary
# integration measure ds would need to be scaled using Nanson's
# formula, but, here, the outflow boundary is fixed so we
# can directly use the ds measure corresponding to the
# reference domain.

# Full fluid residual
res_f = resGal_f + resSUPG_f + resLSIC_f + resOutflow_f
```

```
# Residual of fluid--structure coupled problem:
res_fs = res_f + res_s
Dres_fs = derivative(res_fs, w)

####### Nonlinear solver setup #######

# Nonlinear solver parameters; set relative tolerances
# for subproblems tighter than tolerance for coupled problem,
# to prevent stagnation.
REL_TOL_FSM = 1e-3
REL_TOL_FS = REL_TOL_M = REL_TOL_FSM*1e-1
MAX_ITERS_FSM = 100
MAX_ITERS_M = 100
MAX_ITERS_FS = 100

# Set up nonlinear problem for mesh motion:
problem_m = NonlinearVariationalProblem(res_m, uhat,
                                        bcs_m, Dres_m)
solver_m = NonlinearVariationalSolver(problem_m)
solver_m.parameters['newton_solver']\
                   ['maximum_iterations'] = MAX_ITERS_M
solver_m.parameters['newton_solver']\
                   ['relative_tolerance'] = REL_TOL_M

# Create variational problem and solver for
# the fluid--structure problem:
problem_fs = NonlinearVariationalProblem(res_fs, w,
                                         bcs_fs, Dres_fs)
solver_fs = NonlinearVariationalSolver(problem_fs)
solver_fs.parameters['newton_solver']\
                    ['maximum_iterations'] = MAX_ITERS_FS
solver_fs.parameters['newton_solver']\
                    ['relative_tolerance'] = REL_TOL_FS

####### Time stepping loop #######

# Create files for storing solution:
vfile = File("results/velocity.pvd")
pfile = File("results/pressure.pvd")
mfile = File("results/mesh.pvd")

# Initialize time and step counter.
t = float(Dt)
count = 0
```

```
# Prevent divide-by-zero in relative residual on first
# iteration of first step.
for bc in bcs_fs:
    bc.apply(w.vector())
while t < TIME_INTERVAL:

    print(80*"=")
    print("  Time step "+str(count+1)+" , t = "+str(t))
    print(80*"=")

    # Use the current time in the inflow BC definition.
    v_in.t = t

    # "Quasi-direct" coupling: the fluid and structure
    # are solved in one system, but the mesh is solved
    # in a separate block.
    for i in range(0,MAX_ITERS_FSM):

        # Check fluid--structure residual on the moved
        # mesh, and terminate iteration if this residual
        # is small:
        res_fs_vec = assemble(res_fs)
        for bc in bcs_fs:
            bc.apply(res_fs_vec,w.vector())
        res_norm = norm(res_fs_vec)
        if(i==0):
            res_norm0 = res_norm
        res_rel = res_norm/res_norm0
        print(80*"*")
        print("  Coupling iteration: "+str(i+1)
              +" , Relative residual = "+str(res_rel))
        print(80*"*")
        if(res_rel < REL_TOL_FSM):
            break

        # Solve for fluid/structure velocity and
        # pressure at current time:
        solver_fs.solve()

        # Update Function in V to be used in mesh
        # motion BC.  (There are workarounds to avoid
        # this projection (which requires a linear
        # solve), but projection is most concise for
        # illustration.)
        u_func.assign(project(u,V))
        # Mesh motion problem; updates uhat at current
        # time level:
        solver_m.solve()

    # Extract solutions:
    (v, p) = w.split()

    # Save to file
    v.rename("v","v")
    p.rename("p","p")
    uhat.rename("u","u")
    vfile << v
    pfile << p
    mfile << uhat

    # Move to next time step:
    uhat_old.assign(uhat)
    w_old.assign(w)
    count += 1
    t += float(Dt)
```

Note that, to visualize the results on the deforming spatial domain Ω_x in ParaView, the vector field in `"results/mesh.pvd"` must be applied as a Warp by Vector filter; the mesh used by FEniCS is of Ω_y, and does not deform. See notes in Chapter 3 for further information on combining results from different files in ParaView visualizations.

Bibliography

[1] Y. Bazilevs, K. Takizawa, and T. E. Tezduyar. *Computational Fluid–Structure Interaction: Methods and Applications*. Wiley, Chichester, 2013.

[2] A. Logg and G. N. Wells. DOLFIN: Automated finite element computing. *ACM Trans. Math. Softw.*, 37(2):20:1–20:28, April 2010.

[3] MAE 207 - FEA for coupled problems. `https://github.com/david-kamensky/mae-207-fea-for-coupled-problems`, Accessed November 2023. Source code for examples and exercises.

[4] T. J. R Hughes. *The Finite Element Method. Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[5] L. C. Evans. *Partial Differential Equations (Graduate Studies in Mathematics, Vol. 19)*. American Mathematical Society, Providence, Rhode Island, 2002.

[6] W. Rudin. *Principles of Mathematical Analysis, Third Edition*. McGraw–Hill, Inc., 1976.

[7] J. T. Oden and L. F. Demkowicz. *Applied Functional Analysis, Second Edition*. CRC Press, Boca Raton, FL, USA, 2010.

[8] E. Kreyszig. *Introductory Functional Analysis with Applications*. Wiley Classics Library. Wiley, 1989.

[9] T. Arbogast and J. L. Bona. Methods of applied mathematics (2008 corrected version). `https://web.ma.utexas.edu/users/arbogast/appMath08c.pdf`, Accessed December 2019.

[10] R. A. Adams. *Sobolev Spaces*. Academic Press, New York, 1975.

[11] L. F. Demkowicz. Lecture notes on energy spaces. Technical Report 18-13, ICES, UT Austin, 2018.

[12] T. J. R. Hughes, G. Engel, L. Mazzei, and M. G. Larson. The continuous Galerkin method is locally conservative. *Journal of Computational Physics*, 163(2):467–488, 2000.

[13] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Dover Books on Mathematics Series. Dover Publications, Incorporated, 2012.

[14] E. B. Becker, G. F. Carey, and J. T. Oden. *Finite Elements: I, An Introduction*. Prentice Hall Publishing Company, Englewood, NJ, 1981.

[15] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method*. Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[16] M. S. Gockenbach. *Understanding And Implementing the Finite Element Method*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[17] L. F. Demkowicz. Babuška $\iff$ Brezzi? Technical Report 06-08, ICES, UT Austin, 2006.

[18] B. Cockburn, G. E. Karniadakis, and C.-W. Shu. *Discontinuous Galerkin Methods: Theory, Computation and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[19] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley, Chichester, 2009.

[20] D. N. Arnold. *Finite Element Exterior Calculus*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2018.

[21] A. Logg, K.-A. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[22] FEniCS homepage. `https://fenicsproject.org/`, Accessed December 2019.

[23] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Trans. Math. Softw.*, 40(2):9:1–9:37, March 2014.

[24] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. Mcrae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly. Firedrake: Automating the finite element method by composing abstractions. *ACM Trans. Math. Softw.*, 43(3), December 2016.

[25] R. C. Kirby and A. Logg. A compiler for variational forms. *ACM Trans. Math. Softw.*, 32(3):417–444, September 2006.

[26] R. C. Kirby. Algorithm 839: FIAT, a new paradigm for computing finite element basis functions. *ACM Trans. Math. Softw.*, 30(4):502–516, December 2004.

[27] FEniCS roadmap 2019–2020. `https://fenicsproject.org/fenics-project-roadmap-2019/`, Accessed December 2019.

[28] FEniCS download/installation instructions. `https://fenicsproject.org/download/`, Accessed December 2019.

[29] FEniCS documentation. `https://fenicsproject.org/documentation/`, Accessed December 2019.

[30] Jeremy Bleyer. Numerical tours of computational mechanics with FEniCS. 2018.

[31] B. E. Abali. *Computational Reality*. Springer Singapore, 2017.

[32] FEniCS discourse group. `https://fenicsproject.discourse.group`, Accessed December 2019.

[33] FEniCS slack channel. `https://fenicsproject.slack.com/`, Accessed December 2019.

[34] Gmsh. Automatic 3D tetrahedral mesh generator. `http://gmsh.info/`. Main developers: C. Geuzaine and J.-F. Remacle.

[35] pygmsh source code. `https://github.com/nschloe/pygmsh`, Accessed December 2019.

[36] meshio source code. `https://github.com/nschloe/meshio`, Accessed December 2019.

[37] D. Kamensky and Y. Bazilevs. tIGAr: Automating isogeometric analysis with FEniCS. *Computer Methods in Applied Mechanics and Engineering*, 344:477–498, 2019.

[38] Kitware Inc. Paraview. `https://www.paraview.org/`.

[39] Y. Bazilevs, V. M. Calo, T. J. R. Hughes, and Y. Zhang. Isogeometric fluid–structure interaction: theory, algorithms, and computations. *Computational Mechanics*, 43:3–37, 2008. Preprint: `https://www.oden.utexas.edu/media/reports/2008/0816.pdf`.

[40] J. E. Marsden and T. J. R. Hughes. *Mathematical Foundations of Elasticity*. Dover Civil and Mechanical Engineering Series. Dover, 1994.

[41] F. Brezzi. On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 8(R2):129–151, 1974.

[42] Ivo Babuška. Error-bounds for finite element method. *Numerische Mathematik*, 16(4):322–333, 1971.

[43] D. S. Malkus. Eigenproblems associated with the discrete LBB condition for incompressible finite elements. *International Journal of Engineering Science*, 19(10):1299–1310, 1981.

[44] D. Boffi, F. Brezzi, and M. Fortin. Finite elements for the Stokes problem. In D. Boffi and L. Gastaldi, editors, *Mixed Finite Elements, Compatibility Conditions, and Applications*, Lecture Notes in Mathematics, pages 45–100. Springer-Verlag Berlin Heidelberg, 2008.

[45] J. A. Evans. *Divergence-free B-spline Discretizations for Viscous Incompressible Flows*. Ph.D. thesis, University of Texas at Austin, Austin, Texas, United States, 2011.

[46] D. Kamensky, M.-C. Hsu, Y. Yu, J. A. Evans, M. S. Sacks, and T. J. R. Hughes. Immersogeometric cardiovascular fluid–structure interaction analysis using divergence-conforming B-splines. *Computer Methods in Applied Mechanics and Engineering*, 2016.

[47] H. Casquero, Y. J. Zhang, C. Bona-Casas, L. Dalcin, and H. Gomez. Non-body-fitted fluid–structure interaction: Divergence-conforming B-splines, fully-implicit dynamics, and variational formulation. *Journal of Computational Physics*, 374:625–653, 2018.

[48] H. Casquero, C. Bona-Casas, D. Toshniwal, T. J. R. Hughes, H. Gomez, and Y. J. Zhang. The divergence-conforming immersed boundary method: Application to vesicle and capsule dynamics. *Journal of Computational Physics*, 425:109872, 2021.

[49] `https://github.com/david-kamensky/tIGAr`. tIGAr source code.

[50] I. Harari and T. J. R. Hughes. What are C and h?: Inequalities for the analysis and design of finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 97:157–192, 1992.

[51] T. J. R. Hughes, L. P. Franca, and M. Balestra. A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška–Brezzi condition: A stable Petrov–Galerkin formulation of the Stokes problem accommodating equal-order interpolations. *Computer Methods in Applied Mechanics and Engineering*, 59:85–99, 1986.

[52] A. N. Brooks and T. J. R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32:199–259, 1982.

[53] T. J. R. Hughes, G. R. Feijóo, L. Mazzei, and J. B. Quincy. The variational multiscale method–A paradigm for computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 166:3–24, 1998.

[54] T. J. R. Hughes and G. Sangalli. Variational multiscale analysis: the fine-scale Green's function, projection, optimization, localization, and stabilized methods. *SIAM Journal on Numerical Analysis*, 45(2):539–557, 2007.

[55] K. E. Jansen, S. S. Collis, C. Whiting, and F. Shakib. A better consistency for low-order stabilized finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 174(1):153–170, 1999.

[56] L. P. Franca, S. L. Frey, and T. J. R. Hughes. Stabilized finite element methods: I. Application to the advective-diffusive model. *Computer Methods in Applied Mechanics and Engineering*, 95(2):253–276, 1992.

[57] G. Sangalli. Quasi optimality of the SUPG method for the one-dimensional advection-diffusion problem. *SIAM Journal on Numerical Analysis*, 41(4):1528–1542, 2003.

[58] Y. Bazilevs, V. M. Calo, J. A. Cottrel, T. J. R. Hughes, A. Reali, and G. Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 197:173–201, 2007.

[59] T. J. R. Hughes, L. Mazzei, and K. E. Jansen. Large-eddy simulation and the variational multiscale method. *Computing and Visualization in Science*, 3:47–59, 2000.

[60] J. Hoffman and C. Johnson. A new approach to computational turbulence modeling. *Computer Methods in Applied Mechanics and Engineering*, 195(23):2865–2880, 2006. Incompressible CFD.

[61] Y. Bazilevs, C. Michler, V. M. Calo, and T. J. R. Hughes. Isogeometric variational multiscale modeling of wall-bounded turbulent flows with weakly enforced boundary conditions on unstretched meshes. *Computer Methods in Applied Mechanics and Engineering*, 199:780–790, 2010.

[62] Y. Bazilevs and I. Akkerman. Large eddy simulation of turbulent Taylor–Couette flow using isogeometric analysis and the residual–based variational multiscale method. *Journal of Computational Physics*, 229:3402–3414, 2010.

[63] Y. Bazilevs, M.-C. Hsu, I. Akkerman, S. Wright, K. Takizawa, B. Henicke, T. Spielman, and T. E. Tezduyar. 3D simulation of wind turbine rotors at full scale. Part I: Geometry modeling and aerodynamics. *International Journal for Numerical Methods in Fluids*, 65:207–235, 2011.

[64] Y. Bazilevs, M.-C. Hsu, J. Kiendl, R. Wüchner, and K.-U. Bletzinger. 3D simulation of wind turbine rotors at full scale. Part II: Fluid–structure interaction modeling with composite blades. *International Journal for Numerical Methods in Fluids*, 65:236–253, 2011.

[65] `https://github.com/david-kamensky/VarMINT`. VarMINT source code.

[66] Q. Zhu and J. Yan. A moving-domain CFD solver in FEniCS with applications to tidal turbine simulations in turbulent flows. *Computers & Mathematics with Applications*, 2019.

[67] `https://github.com/QimingZhu1992/IlliniFlow`. IlliniFlow source code.

[68] M.F.P. ten Eikelder, Y. Bazilevs, and I. Akkerman. A theoretical framework for discontinuity capturing: Joining variational multiscale analysis and variation entropy theory. *Computer Methods in Applied Mechanics and Engineering*, page 112664, 2019.

[69] S. K. F. Stoter, M. F. P. ten Eikelder, F. de Prenter, I. Akkerman, E. H. van Brummelen, C. V. Verhoosel, and D. Schillinger. Unification of variational multiscale analysis and Nitsche's method, and a resulting boundary layer fine-scale model, 2020.

[70] J. Hoffman, J. Jansson, R. Vilela de Abreu, N. Cem Degirmenci, N. Jansson, K. Müller, M. Nazarov, and J. H. Spühler. Unicorn: Parallel adaptive finite element simulation of turbulent flow and fluid–structure interaction for deforming domains and complex geometry. *Computers & Fluids*, 80:310–319, 2013. Selected contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011.

[71] J. Jansson, E. Krishnasamy, M. Leoni, N. Jansson, and J. Hoffman. Adaptive direct FEM simulation for HiLiftPW-3. In *3rd High Lift Prediction Workshop*, 2017.

[72] L. P. Franca and S. L. Frey. Stabilized finite element methods: II. the incompressible Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 99(2):209–233, 1992.

[73] T. J. R. Hughes, G. Scovazzi, and T. E. Tezduyar. Stabilized methods for compressible flows. *Journal of Scientific Computing*, 43(3):343–368, 2010.

[74] Y. Bazilevs, M.-C. Hsu, and M. A. Scott. Isogeometric fluid–structure interaction analysis with emphasis on non-matching discretizations, and with application to wind turbines. *Computer Methods in Applied Mechanics and Engineering*, 249–252:28–41, 2012.

[75] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, and S. Mittal. Parallel finite-element computation of 3D flows. *Computer*, 26(10):27–36, 1993.

[76] A. A. Johnson and T. E. Tezduyar. Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces. *Computer Methods in Applied Mechanics and Engineering*, 119:73–94, 1994.

[77] K. Stein, T. Tezduyar, and R. Benney. Mesh moving techniques for fluid–structure interactions with large displacements. *Journal of Applied Mechanics*, 70:58–63, 2003.

[78] K. Stein, T. E. Tezduyar, and R. Benney. Automatic mesh update with the solid-extension mesh moving technique. *Computer Methods in Applied Mechanics and Engineering*, 193:2019–2032, 2004.

[79] H. J. C. Barbosa and T. J. R. Hughes. The finite element method with Lagrange multipliers on the boundary: circumventing the Babuška-Brezzi condition. *Computer Methods in Applied Mechanics and Engineering*, 85(1):109–128, 1991.

[80] R. Stenberg. On some techniques for approximating boundary conditions in the finite element method. *Journal of Computational and Applied Mathematics*, 63(1-3):139–148, 1995.

[81] J. Nitsche. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 36:9–15, 1971.

[82] J. Pitkäranta. Local stability conditions for the Babuška method of Lagrange multipliers. *Mathematics of Computation*, 35(152):1113–1129, 1980.

[83] D. Schillinger, I. Harari, M.-C. Hsu, D. Kamensky, S. K. F. Stoter, Y. Yu, and Y. Zhao. The non-symmetric Nitsche method for the parameter-free imposition of weak boundary and coupling conditions in immersed finite elements. *Computer Methods in Applied Mechanics and Engineering*, 309:625–652, 2016.

[84] T. J. R. Hughes and G. N. Wells. Conservation properties for the Galerkin and stabilised forms of the advection–diffusion and incompressible Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 194(9):1141–1159, 2005.

[85] C. S. Peskin. Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10(2):252–271, 1972.

[86] C. S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002.

[87] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annual Review of Fluid Mechanics*, 37:239–261, 2005.

[88] F. Sotiropoulos and X. Yang. Immersed boundary methods for simulating fluid–structure interaction. *Progress in Aerospace Sciences*, 65:1–21, 2014.

[89] B. E. Griffith and N. A. Patankar. Immersed methods for fluid–structure interaction. *Annual Review of Fluid Mechanics*, 52(1):null, 2020.

[90] A. Johansson, B. Kehlet, M. G. Larson, and A. Logg. Multimesh finite element methods: Solving PDEs on multiple intersecting meshes. *Computer Methods in Applied Mechanics and Engineering*, 343:672–689, 2019.

[91] J. Parvizian, A. Düster, and E. Rank. Finite cell method. *Computational Mechanics*, 41(1):121–133, 2007.

[92] D. Schillinger and M. Ruess. The finite cell method: A review in the context of higher-order structural analysis of CAD and image-based geometric models. *Archives of Computational Methods in Engineering*, 22(3):391–455, 2015.

[93] N. Zander, T. Bog, M. Elhaddad, R. Espinoza, H. Hu, A. Joly, C. Wu, P. Zerbe, A. Düster, S. Kollmannsberger, J. Parvizian, M. Ruess, D. Schillinger, and E. Rank. FCMLab: A finite cell research toolbox for MATLAB. *Advances in Engineering Software*, 74:49–63, 2014.

[94] A. Stavrev. The role of higher-order geometry approximation and accurate quadrature in nurbs based immersed boundary methods. Master's thesis, Technische Universität München, Munich, Germany, 2012.

[95] J. A. Nitsche and A. H. Schatz. Interior estimates for Ritz–Galerkin methods. *Mathematics of Computation*, 28(128):937–958, 1974.

[96] S. Bertoluzza, M. Ismail, and B. Maury. Analysis of the fully discrete fat boundary method. *Numerische Mathematik*, 118(1):49–77, 2011.

[97] F. Xu, D. Schillinger, D. Kamensky, V. Varduhn, C. Wang, and M.-C. Hsu. The tetrahedral finite cell method for fluids: Immersogeometric analysis of turbulent flow around complex geometries. *Computers & Fluids*, 141:135–154, 2016. Advances in Fluid-Structure Interaction.

[98] M.-C. Hsu, C. Wang, F. Xu, A. J. Herrema, and A. Krishnamurthy. Direct immersogeometric fluid flow analysis using B-rep CAD models. *Computer Aided Geometric Design*, 43:143–158, 2016. Geometric Modeling and Processing 2016.

[99] C. Kadapa, W. G. Dettmer, and D. Perić. A stabilised immersed framework on hierarchical B-spline grids for fluid-flexible structure interaction with solid–solid contact. 335:472–489, 2018.

[100] K. Saurabh, B. Gao, M. Fernando, So.e Xu, B. Khara, M. A. Khanwale, M.-C. Hsu, A. Krishnamurthy, H. Sundar, and B. Ganapathysubramanian. Industrial scale large eddy simulations (LES) with adaptive octree meshes using immersogeometric analysis, 2020.

[101] R. Glowinski, T.-W. Pan, and J. Periaux. A fictitious domain method for Dirichlet problem and applications. *Computer Methods in Applied Mechanics and Engineering*, 111(3):283–303, 1994.

[102] F.P. T. Baaijens. A fictitious domain/mortar element method for fluid–structure interaction. *International Journal for Numerical Methods in Fluids*, 35(7):743–761, 2001.

[103] J. de Hart, G. W. M. Peters, P. J. G. Schreurs, and F. P. T. Baaijens. A three-dimensional computational analysis of fluid–structure interaction in the aortic valve. *Journal of Biomechanics*, 36(1):103–112, 2003.

[104] D. Kamensky, M.-C. Hsu, D. Schillinger, J. A. Evans, A. Aggarwal, Y. Bazilevs, M. S. Sacks, and T. J. R. Hughes. An immersogeometric variational framework for fluid–structure interaction: Application to bioprosthetic heart valves. *Computer Methods in Applied Mechanics and Engineering*, 284:1005–1053, 2015.

[105] M.-C. Hsu, D. Kamensky, F. Xu, J. Kiendl, C. Wang, M. C. H. Wu, J. Mineroff, A. Reali, Y. Bazilevs, and M. S. Sacks. Dynamic and fluid–structure interaction simulations of bioprosthetic heart valves using parametric design with T-splines and Fung-type material models. *Computational Mechanics*, 55:1211–1225, 2015.

[106] D. Kamensky, J. A. Evans, and M.-C. Hsu. Stability and conservation properties of collocated constraints in immersogeometric fluid-thin structure interaction analysis. *Communications in Computational Physics*, 18:1147–1180, 2015.

[107] D. Kamensky, J. A. Evans, M.-C. Hsu, and Y. Bazilevs. Projection-based stabilization of interface Lagrange multipliers in immersogeometric fluid–thin structure interaction analysis, with application to heart valve modeling. *Computers & Mathematics with Applications*, 74(9):2068–2088, 2017. Advances in Mathematics of Finite Elements, honoring 90th birthday of Ivo Babuška.

[108] E. L. Johnson, M. C. H. Wu, F. Xu, N. M. Wiese, M. R. Rajanna, A. J. Herrema, B. Ganapathysubramanian, T. J. R. Hughes, M. S. Sacks, and M.-C. Hsu. Thinner biological tissues induce leaflet flutter in aortic heart valve replacements. *Proceedings of the National Academy of Sciences*, 117(32):19007–19016, 2020.

[109] Y. Yu, D. Kamensky, M.-C. Hsu, X. Y. Lu, Y. Bazilevs, and T. J. R. Hughes. Error estimates for projection-based dynamic augmented Lagrangian boundary condition enforcement, with application to fluid–structure interaction. *Mathematical Models and Methods in Applied Sciences*, 28:2457–2509, 2018.

[110] D. Kamensky. Open-source immersogeometric analysis of fluid–structure interaction using FEniCS and tIGAr. *Computers & Mathematics with Applications*, 2020. In press.

[111] E. H. van Brummelen and R. de Borst. On the nonnormality of subiteration for a fluid-structure-interaction problem. *SIAM Journal on Scientific Computing*, 27(2):599–621, 2005.

[112] E. H. van Brummelen. Added mass effects of compressible and incompressible flows in fluid–structure interaction. *Journal of Applied Mechanics*, 76:021206, 2009.

[113] U. Kuttler and W. A. Wall. Fixed-point fluid–structure interaction solvers with dynamic relaxation. *Computational Mechanics*, 43:61–72, 2008.

[114] M.D. Greenberg. *Foundations of Applied Mathematics*. Dover Publications, 2013.

[115] G. A. Holzapfel. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*. Wiley, Chichester, 2000.